Understanding Bloom Filter Intersection for Lazy Address-Set Disambiguation

Mark C. Jeffrey and J. Gregory Steffan
Department of Electrical and Computer Engineering
University of Toronto
{markj,steffan}@eecg.toronto.edu

ABSTRACT

A Bloom filter is a probabilistic bit-array-based set representation that has recently been applied to address-set *disambiguation* in systems that ease the burden of parallel programming. However, many of these systems intersect the Bloom filter bit-arrays to approximate address-set intersection and decide set disjointness. This is in contrast with the conventional and well-studied approach of making individual membership queries into the Bloom filter. In this paper we present much-needed probabilistic models for the unconventional application of testing set disjointness using Bloom filters. Consequently, we demonstrate that intersecting Bloom filters requires substantially larger bit-arrays to provide the same probability of *false set-overlap* as querying into the bit-array. For when intersection is unavoidable, we prove that partitioned Bloom filters require less space than unpartitioned. Finally, we show that for Bloom filters with a single hash function, surprisingly, intersection and querying share the same probability of false set-overlap.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms

Design, Performance, Theory

Keywords

Bloom filters, signatures, set intersection, address-set disambiguation, transactional memory, thread-level speculation, parallelism

1. INTRODUCTION

The over-arching challenge for parallel programming stems from detecting and managing data access conflicts between parallel threads, since they can lead to invalid data and incorrect execution when improperly handled. A variety of programming models and debug tools have hence been proposed to augment locking, a conventional form of managing potential conflicts. Progress has been made in tools for finding, replaying, and avoiding concurrency bugs [18] that may result when a programmer: (i) fails to synchronize accesses to a mutable shared variable (i.e., a data race) [28]; or (ii) incorrectly reasons about atomicity, failing to enclose a set of memory accesses in a critical section (i.e., an atomicity violation) [17, 18]. Debugging in a concurrent environment is made even more challenging as the manifestation of these bugs depends on the non-deterministic interleavings of threads. Several debugging systems thus focus on deterministically replaying concurrency bugs to find their source [13,25,38]. Despite thorough testing, some bugs still make it to deployment, motivating dynamic avoidance of concurrency bugs [17]. Beyond debugging, Transactional Memory (TM) [12] and Thread-Level Speculation (TLS) [10, 15, 37] have emerged as methods of more automatically managing data access conflicts for the programmer. TM allows potentially conflicting transactions to execute concurrently, where the underlying system tracks memory accesses and detects and handles data conflicts. TLS divides a legacy sequential program into ordered speculative threads that are executed optimistically in parallel, also via an underlying system of detecting and recovering from data access conflicts.

All of these recently-proposed programming models and tools require a means of conflict detection (CD) that disambiguates streams of concurrent memory address accesses to find unsafe access interleavings (conflicts). Read- and write-sets accumulate the memory addresses read or written over *epochs* of instructions as defined by the application—including synchronization points, race-free episodes, transactions, or "chunks" of sequentially-consistent instructions [5]. In general, a conflict results when a memory address appears in the write-set of one thread and the read- or write-set of another thread. Two schedules determine when conflicts are detected: *eagerly* at the time of memory access (e.g., by checking coherence messages [26, 41]), or *lazily* at the end of an epoch.

1.1 Address-Set Disambiguation Using Bloom Filters

Given this demand for runtime address-set operations, Bloom filters [1] have emerged as the address-set representation of choice for many systems in hardware TM [6, 16, 21, 32–35, 39–42], software TM [8, 19, 30, 36], TLS [6, 11], and concurrency debugging tools [13, 17, 18, 25, 28, 31, 43]. These approximate set representations provide address membership queries and set insertion in constant time, while operating on a compact, static-length bit-array. To track an address-set using a Bloom filter, addresses are hash-encoded into the large bit-array, and each thread maintains a distinct Bloom filter for each of the read- and write-sets over the course of each execution epoch. The bit-array length is designer-tunable, but the filter suffers increasing inaccuracy as length decreases; hence space and time requirements must be balanced with an acceptable probability that set membership tests falsely accept a non-member. In parallelization systems, these Bloom filter *false positives* force unnecessary conflicts among epochs, but pose no threat to correctness—e.g., their impact is limited to the re-execution of a transaction or epoch, or a false concurrency bug reported to the programmer. Hardware systems leverage Bloom filters to represent unbounded sets using statically-sized registers, and software systems benefit from fast set operations. The main concern for designers of these parallelization systems is to size the Bloom filters appropriately to achieve an acceptable false positive rate.

Despite the popularity of using Bloom filters for address-set disambiguation, few analytical models have been developed for these use-scenarios: for most recent work, the bit-arrays in Bloom filters are sized via time-consuming design space exploration, where the false positive rates are determined empirically. Eager systems use Bloom filter membership queries to detect conflicts, and the resulting false positives of individual queries follow a well-understood probability distribution [1, 2, 7, 33]. Configuring Bloom filters for individual queries therefore requires tuning only the Bloom filter length and number of hash functions, and can be guided by the known analytical model.

1.2 Needed: Analytical Models for Lazy Applications of Address-Set Bloom Filters

In contrast with eager systems, lazy systems disambiguate finalized address-sets at the end of epochs[1], affording designers more flexibility but an expanded design space to explore. With finalized Bloom filters, there exist three different methods of determining whether address-sets are disjoint, or *deciding set disjointness*. The first method is intuitive: test every address of one set for membership in the Bloom filter of the other set [21]—we call this method *queue-of-queries*. For both the second and third approach, rather than serially querying many addresses, Bloom filters are quickly *intersected* and the result is analyzed to determine whether the input sets are disjoint [5, 6, 8, 17, 19, 25, 28, 30, 36, 43]. For the second approach, Bloom filter intersection approximates set intersection by performing the bit-wise `AND` of two bit-arrays, but has lower resolution (i.e., a greater probability of false conflict) than the corresponding series of queries. The third approach partitions the two bit-arrays, and the partitions are pairwise intersected. The third approach is hence called *partitioned*, while the second approach is unpartitioned—sometimes referred to as a *true* Bloom filter [32, 33]. When the two input sets are disjoint, each disambiguation method might return a *false set-overlap*. In this context, the statistical properties of the three set-intersecting approximations, to the best of our knowledge, have neither been studied analytically nor conclusively compared in prior work.

[1]There also exist systems where finalized address-sets of a committing epoch are compared with growing address-sets of in-flight epochs. We continue to use the term "finalized set," but this could alternatively be interpreted as "nontrivial set."

In this paper we provide system designers with a new analytical model of the *probability of false set-overlap* for address-set Bloom filter intersection. We conclusively show which bit-array configuration admits fewer false positives, and prove that to achieve equivalent probability of false set-overlap, intersection-based usage requires Bloom filters that are at least a factor of the square root of set cardinality larger than query-based usage: for example, our models suggest that a change from unpartitioned Bloom filters to 2- or 4-way partitioning of the bit-array will yield considerable reduction of false conflicts in a number of existing parallelization systems [30, 36]. These results also reveal that for set-overlap-testing intersect-based schemes, the query-based approach should still receive serious consideration as an alternative, despite its time complexity.

Related Work: Prior work on address-set Bloom filters (a.k.a. *signatures* [6]) has optimized false positives in membership queries, but has not focused on Bloom filter intersection in particular. The work includes evaluating the impact of hash function families and parallel access to the bit-array partitions [33], optimizing the complexity of hash functions for a fixed false positive rate [42], application-specific address hashing [16], and exploiting the locality of an address stream [32]. The database community has applied Bloom filter intersection to accelerate relational join operations: approximating set intersection, and subsequently performing membership queries to the remaining bits [20]. Estimation of join cardinality has also benefited from this fast intersection [3, 27, 29]. However, unlike address-set disambiguation, database applications generally do not strive for the intersection result to be an empty set. Our work builds on these studies of Bloom filter intersection, with a focus on address-set disambiguation, and hence targeting intersections that return empty sets in the ideal case.

Contributions: This paper makes the following contributions: (i) we derive the probability distributions of false set-overlaps between two address-sets, for each of the three ways Bloom filters are applied in lazy address disambiguation; (ii) we prove that the partitioned Bloom filter configuration statistically induces fewer false conflicts than the unpartitioned configuration; (iii) we prove that for equivalent probability of false set-overlap in Bloom filters, intersection exceeds a space requirement that is larger than querying by a factor of the square root of set cardinality; (iv) we observe that, for the special case of one-way hashing, Bloom filter querying and intersection remarkably share an equivalent probability of false set-overlap.

2. BLOOM FILTERS

This section gives a brief background on the relevant aspects of Bloom filters [1]. For preliminary notation, let $[N]$ denote the set $\{1, \dots, N\}$. A Bloom filter compactly represents a set $S = \{x_1, x_2, \dots, x_n\}$ of n elements from some universe U. The filter is a bit-array of m bits indexed by a hash function tuple of k mutually-independent hash

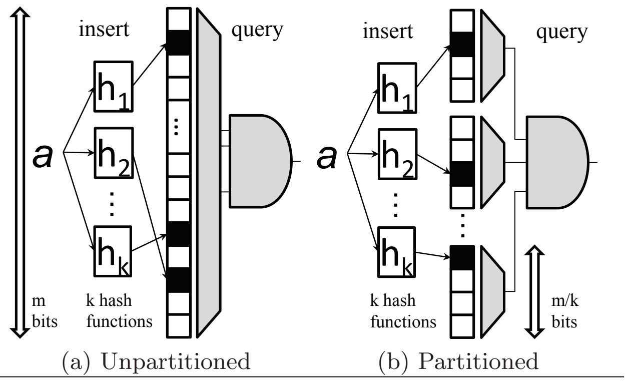

Figure 1: Bloom filter insertion and querying of address a for (a) an unpartitioned and (b) partitioned Bloom filter. In both cases, the filter has length m bits, and a tuple of k truly random hash functions. Addresses are inserted by asserting the bits indexed by the k hash values (dark boxes). A query accepts an address as a member of the set iff all k indexed bits are set to 1. Inspiration for figure is from [32].

functions $h(x) = (h_1(x), \ldots, h_k(x))$, supporting operations such as element insertion, membership queries, set union, and set intersection. Two configurations of bit indexing are widely used, called unpartitioned (or "true") and partitioned. The k-tuple hash function of an unpartitioned Bloom filter uniformly maps to the entire m-bit range of the bit-array, $h : U \to [m]^k$. In contrast, a partitioned Bloom filter distinguishes k disjoint sub-arrays of the filter, with each hash tuple value uniformly mapping to an integer $\frac{m}{k}$-bit partitioned range, $h : U \to \left[\frac{m}{k}\right]^k$ [24]. In the context of address disambiguation, S is a set of memory addresses from a v-bit address space (or universe): $S \subset U = \{0, 1, \ldots, 2^v - 1\}$.

Figure 1 illustrates the individual element operations on a Bloom filter. To initialize an empty set, all bits of the array are set to 0. Each element $x \in S$ is subsequently inserted in the filter by asserting the k bits indexed by each hash of x; the $h_i(x)$'th bit is set to 1 for $1 \leq i \leq k$. We denote the Bloom filter representation of S as $BF(S)$, which corresponds to the set of asserted bits after inserting all $x \in S$ [9, 32]. With a fully-constructed Bloom filter for S, an address $y \in U$ can be quickly tested for membership in S. The membership query accepts $y \in S$ if all of the $h_i(y)$'th bits of the array are 1, and otherwise indicates $y \notin S$.

For the remainder of this paper, we assume that the bit-array length, m, and number of hash functions, k, are constant. Let $\mathcal{H}_m = \{h | h : U \to [m]^k\}$ be the set of all hash functions mapping from the address space to the full m-bit range of a partitioned Bloom filter. Similarly, let $\mathcal{H}_{\frac{m}{k}} = \{h | h : U \to [\frac{m}{k}]^k\}$ represent the set of all hash functions from the address space to the $\frac{m}{k}$-bit sub-arrays of a partitioned Bloom filter.

2.1 Bloom Filters for Membership Queries

By encoding elements of a large universe into a compact bit-array, there is a small probability that an element y (that is not in S) has collisions on each of its k hashes with some elements in the set. Both hash *aliasing* and filter *density* (fraction of asserted bits) can lead to membership queries being falsely accepted, and so the Bloom filter actually represents a superset of the original address set: $S \subseteq BF(S)$. In essence, the membership query "is y in S?" is not answered by no or yes, but rather no or *maybe*. The following definition formalizes this notion.

DEFINITION 1. *Let $S = \{x_1, \ldots, x_n\} \subset U$ be represented by an m-bit Bloom filter, $BF(S)$, using the k-tuple hash function $h \in (\mathcal{H}_m \cup \mathcal{H}_{\frac{m}{k}})$. When testing some element $y \notin S$ for membership in S, we define the* false positive *predicate $\mathrm{FP}_\in(S, y, h)$ to be true when the query accepts y as a member of $BF(S)$—i.e., when $y \notin S$, but $\forall i \in [k]$, $h_i(y) = h_i(x_j)$ for some $x_j \in S$.*

This definition describes the false positive event for both unpartitioned and partitioned Bloom filters, as $h \in (\mathcal{H}_m \cup \mathcal{H}_{\frac{m}{k}})$; however, throughout the paper we will specify the Bloom filter indexing via conditioning on h. The probability of false positives for a Bloom filter is well understood and estimated in a straightforward fashion [1, 3, 32]—the reader is directed to prior work for a formal proof. Assuming the partitioned Bloom filter indexing scheme, the distribution is as follows:

LEMMA 1. [1, 24] *Let $h \in \mathcal{H}_{\frac{m}{k}}$ be a truly random k-tuple hash function. For any fixed set $S \subset U$ and element $y \notin S$, the probability that y is accepted in a membership query of partitioned $BF(S)$ is*

$$\Pr\left[\mathrm{FP}_\in(S, y, h) \mid h \in \mathcal{H}_{\frac{m}{k}}\right] = \left(1 - \left(1 - \frac{k}{m}\right)^{|S|}\right)^k. \quad (1)$$

For unpartitioned Bloom filters, $\Pr[\mathrm{FP}_\in(S, y, h) \mid h \in \mathcal{H}_m]$ is less than the result above, since a partitioned filter typically has more asserted bits. Notably, the two distributions asymptotically approach $\left(1 - e^{-\frac{k|S|}{m}}\right)^k$ [3]^2, and the latter approximation is minimized when $k = \frac{m}{|S|} \ln 2$ [3].

2.2 Bloom Filters for Set Intersection

Beyond individual element operations, Bloom filters can be used to perform set union and intersection. In this work we focus on set intersection and its application for deciding address-set disjointness. Let $S_1, S_2 \subset U$ be two sets that are represented by Bloom filters, $BF(S_1)$ and $BF(S_2)$, that use the same m and hash functions. The filter $BF(S_1 \cap S_2)$ is computed by hash-encoding elements of the actual intersection of these sets. The Bloom filter representations of S_1 and S_2 are insufficient to accurately compute $BF(S_1 \cap S_2)$, but their *Bloom filter intersection*, $BF(S_1) \cap BF(S_2)$, is quickly computed by the bit-wise `AND` of their bit-arrays. Bloom filter intersection provides an approximation to set intersection that maintains the original querying property of never returning false negatives [3, 9, 29].

Guo *et al.* quantify the uncertainty in approximating set intersection with Bloom filter intersection. Assuming the unpartitioned Bloom filter configuration, the theorem by Guo is stated as a lemma toward our own contributions; readers are directed to the original work [9] for a proof.

LEMMA 2. [9] *Assuming the same m and random hash function $h \in \mathcal{H}_m$ are used in the Bloom filters of S_1, S_2, and $S_1 \cap S_2$, then $BF(S_1 \cap S_2) = BF(S_1) \cap BF(S_2)$ with probability*

$$(1 - 1/m)^{k^2 \times |S_1 - S_1 \cap S_2| \times |S_2 - S_1 \cap S_2|}.$$

^{2}Since $(1 - k/m)^n \approx e^{-\frac{kn}{m}}$, provided $m > nk$ [3].

Apparently, the asserted Bloom filter bits of a set intersection are not necessarily equivalent to the bits asserted by Bloom filter intersection of the sets; they are equivalent with non-negligible probability.

2.3 Accuracy of the False Positive Rate

Recent work [2, 7] indicates that the "classic" analysis of the Bloom filter that proves the above Lemmas 1 and 2 is optimistic. The result attributed to Bloom (and republished in decades of subsequent work) is in fact a strict lower bound to the correct false positive probability. The new insight by Bose *et al.* and Christensen *et al.* has only focused on unpartitioned Bloom filters; applying their methods to the partitioned configuration, and subsequently repairing the Lemma by Guo *et al.* is left as future work, beyond the scope of this paper. Regardless, the approximation provided by these lemmas is sufficient for this work, as Christensen *et al.* demonstrated that the relative error diminishes with the larger m (≥ 1024 bits) typically used in parallelization systems [5, 6, 8, 13, 18, 21, 28, 32, 33, 36, 41, 42].

3. MODELING BLOOM FILTERS FOR SET DISJOINTNESS

Despite the popularity of Bloom filters in research architectures and tools, there are no previously-proposed probability distributions that model their use in deciding pairwise disjointness of sets, (a.k.a., set disambiguation). In the following sections, we (i) describe how Bloom filters are used to decide address-set disjointness, (ii) describe when they flag false conflicts among epochs, and (iii) we model and prove the probability distributions representing these unfortunate events.

3.1 Methods of Deciding Set Disjointness

This section describes the three methods of deciding set disjointness using Bloom filters: queue-of-queries, unpartitioned intersection, and partitioned intersection. We first motivate a definition of *false set-overlap*, when two disjoint sets appear to have some overlap due to Bloom filter operations. In line with Bloom's original motivation, systems implementing eager conflict detection use Bloom filter membership queries for runtime address-set comparison. At the time of accessing address y, the address is tested for membership in the read or write Bloom filters ($BF(R)$ or $BF(W)$) of other epochs (e.g., by querying incoming coherence requests). There is a probability of a false positive on each query (unnecessarily indicating an address conflict), which is modeled by Lemma 1. Since address conflicts are detected at the granularity of epochs, it becomes apparent that the probability of individual false conflicts is not of interest in parallel programming tools. Instead we wish to know the probability that entire epochs will falsely conflict, such as for lazy conflict detection schemes where the read- and write-sets are finalized. We next define two predicates which relate epoch failures to false set-overlaps.

3.1.1 Queue-of-Queries

Consider the lazy conflict detection scheme of SigTM [21], which maintains a write buffer (W) and read and write Bloom filters ($BF(R)$ and $BF(W)$) for each thread. These sets are finalized at the end of an epoch and otherwise grow monotonically. To detect conflicts at the end of a transaction, the system verifies that every member of the write-set W is not a member of all other threads' read-sets by performing membership queries into the read filters via coherence broadcasts. If any address in the write-set conflicts with the read filter of a remote transaction, the latter transaction is aborted. We use SigTM as a sample model of what we denote as the conventional approach to lazy address-set intersection—executing a *queue-of-queries* into a Bloom filter. Figure 2a illustrates this idea, where the queue of elements is the aforementioned write buffer. Each element of the write buffer (queue) is queried into the Bloom filter of some other epoch, until a conflict is found; otherwise the sets are disjoint. Supposing the two epochs did in fact access independent memory, we say that a false set-overlap occurred if one of the epochs unnecessarily aborted. The following definition formalizes false set-overlap by a queue-of-queries.

Definition 2. *Let $S_1, S_2 \subset U$ be two fixed, disjoint sets, and choose S_1 to be represented by a Bloom filter of m bits and hash function $h \in (\mathcal{H}_m \cup \mathcal{H}_{\frac{m}{k}})$. We define the* false set-overlap by queries *predicate* $\mathrm{FSO}_{\in}(S_1, S_2, h)$ *to be true if, for some $x \in S_2$,* $\mathrm{FP}_{\in}(S_1, x, h)$ *is true.*

This definition describes when two sets would be incorrectly reported as overlapping by the conventional method of using Bloom filters for membership queries. The predicate is defined for either type of bit-indexing by hash functions, since $\mathrm{FP}_{\in}$ of Section 2 is defined for either hash function. In later sections, we will condition on the bit-indexing scheme as necessary.

3.1.2 Intersection: Partitioned and Unpartitioned

Lazy conflict detection must determine whether particular address-sets are disjoint—i.e., to ask "is their intersection empty?" Some researchers have astutely avoided the linear time required for a queue-of-queries by applying Bloom filter intersection to approximate this underlying set intersection task. Independent of the bit-indexing scheme, the bitwise `AND` of two bit-arrays is performed—the time-complexity of which is determined by the amount of available hardware (some researchers [36] reasonably argue that it is constant time).

On the other hand, determining set *emptiness* depends on the bit-indexing scheme. An unpartitioned Bloom filter represents an empty set if and only if all m bits of the bit-array are set to zero. Consider that if a single bit is set, it is possible (though unlikely) that some element is mapped to that same bit by all k hash values, making the filter non-empty. In contrast, partitioned Bloom filters represent an empty set if and only if *at least* one partition is empty, with all m/k bits set to zero [6]. For sufficiency, note that an empty set asserts no bits, such that all k partitions remain zero. For necessity, since inserting one element requires asserting one bit in all partitions, then if at least one partition is empty, it must be that no combination of elements can be represented by that filter—i.e., the filter is empty. Figures 2b and 2c use logic gates to illustrate the use of Bloom filter intersection to test for set-overlap.

The following definition introduces a predicate that identifies false set-overlap via Bloom filter intersection. Due to the difference in empty-set representation, partitioned and unpartitioned filters have differing statistical properties; we

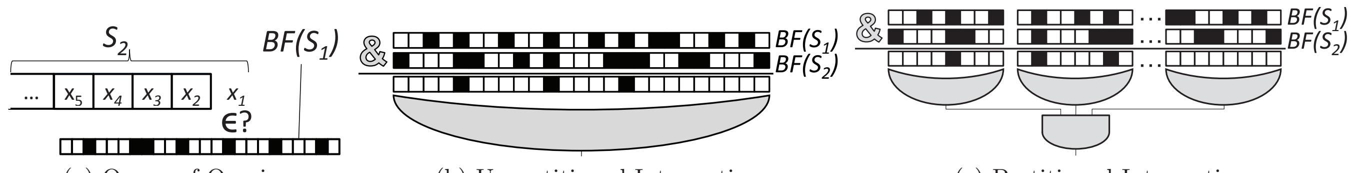

Figure 2: Three methods of testing set-overlap between sets S_1 and S_2: (a) by a queue-of-queries into the Bloom filter of S_1, such that if any element of S_2 matches in $BF(S_1)$, the sets are reported to be non-disjoint; (b) by intersecting two unpartitioned Bloom filters by bitwise `AND`, where any resulting asserted bits indicate non-disjoint sets; (c) by intersecting two partitioned Bloom filters, where an intersection-result consisting of at least one empty partition indicates that the input sets are disjoint.

condition on the choice of hash indexing scheme in the next section.

Definition 3. *Let $S_1, S_2 \subset U$ be two fixed, disjoint sets, each represented by Bloom filters of m bits and hash function $h \in (\mathcal{H}_m \cup \mathcal{H}_{\frac{m}{k}})$. We define the* false set-overlap by Bloom intersection *predicate* $\text{FSO}_\cap(S_1, S_2, h)$ *to be true, if $BF(S_1) \cap BF(S_2) \neq \emptyset$, even though $S_1 \cap S_2 = \emptyset$.*

3.2 Probability of False Set-Overlap

Having defined the conditions for three types of Bloom filter false set-overlap events, we now model their probability distributions. We begin with the probability of false set-overlap by queue-of-queries—using filters as Bloom "intended". Concerning the following theorem, fix two disjoint sets $S_1, S_2 \subset U$. The filter $BF(S_1)$ is m bits long using a truly random hash function of the partitioned bit-indexing scheme: $h \in \mathcal{H}_{\frac{m}{k}}$.

Theorem 1. *A false set-overlap by queries of S_2 into partitioned $BF(S_1)$ is reported with probability*[3]

$$\Pr\left[\text{FSO}_\in(S_1, S_2, h) \mid h \in \mathcal{H}_{\frac{m}{k}}\right] = 1 - \left(1 - \left(1 - \left(1 - \frac{k}{m}\right)^{|S_1|}\right)^k\right)^{|S_2|}. \quad (2)$$

Proof. Consider the contrary, between the two disjoint sets S_1 and S_2, when will a false set-overlap be avoided? Using the Bloom filter representation, the sets are correctly reported disjoint iff $(\forall x \in S_2)(x \notin BF(S_1))$, when every one of the $|S_2|$ unique queries into $BF(S_1)$ does *not* return a false positive. Model these unique queries as a sequence of up to $|S_2|$ Bernoulli trials, where a trial "success" implies a false positive on an individual query. Let random variable N be the number of unique membership queries from S_2 before one is reported a false positive. Thus N follows a geometric distribution, the number of Bernoulli failures before the first success, with probability of success $p = \Pr[\text{FP}_\in(S_1, x, h)]$ for any $x \in S_2$. The two sets are deemed disjoint by Bloom filter queries if all $|S_2|$ trials fail, or if $N \geq |S_2|$. A false set-overlap results if the latter is not true. Thus,

$$\begin{aligned}
&\Pr[\text{FSO}_\in(S_1, S_2, h)] \\
&\quad = \Pr[N < |S_2|] && (3)\\
&\quad = \Pr[N \leq |S_2| - 1] && (4)\\
&\quad = 1 - (1-p)^{|S_2|-1+1} && (5)\\
&\quad = 1 - (1 - \Pr[\text{FP}_\in(S_1, x, h)])^{|S_2|}, x \in S_2 && (6)
\end{aligned}$$

where Eq. (5) substitutes the geometric cumulative distribution function. Conditioning Eq. (6) on h and substituting Eq. (1) gives (2). □

We now state and prove the probability that Bloom filter intersection will flag a false set-overlap[4] (for both unpartitioned and partitioned bit-indexing). Let $S_1, S_2 \subset U$ be disjoint sets. Both are represented by Bloom filters with length m bits, using the same truly random hash function tuple h. The bit-indexing scheme is conditioned in the theorem.

Theorem 2. *A false set-overlap by Bloom filter intersection of unpartitioned $BF(S_1)$ and $BF(S_2)$ is reported with probability*

$$\Pr\left[\text{FSO}_\cap \mid h \in \mathcal{H}_m\right] = 1 - \left(1 - \frac{1}{m}\right)^{k^2|S_1||S_2|}. \quad (7)$$

For partitioned Bloom filters, a false set-overlap is reported with probability

$$\Pr\left[\text{FSO}_\cap \mid h \in \mathcal{H}_{\frac{m}{k}}\right] = \left(1 - \left(1 - \frac{k}{m}\right)^{|S_1||S_2|}\right)^k. \quad (8)$$

Proof. Concerning Eq. (7), Section 3.1.2 argues that intersection of unpartitioned Bloom filters induces a false set-overlap when some bit in the resulting bit-array is nonzero. An all-zero Bloom filter can only be created from an empty set (i.e., $BF(S) = \emptyset \iff S = \emptyset$). Therefore,

$$\begin{aligned}
&\Pr\left[\text{FSO}_\cap \mid h \in \mathcal{H}_m\right] \\
&\quad = \Pr\left[\neg(BF(S_1) \cap BF(S_2) = \emptyset) \mid S_1 \cap S_2 = \emptyset, h \in \mathcal{H}_m\right] && (9)\\
&\quad = 1 - \Pr\left[BF(S_1) \cap BF(S_2) = \emptyset \mid S_1 \cap S_2 = \emptyset, h \in \mathcal{H}_m\right] && (10)
\end{aligned}$$

[3]Building on Lemma 1, this distribution can also be approximated by $1 - \left(1 - \left(1 - e^{-\frac{k|S_1|}{m}}\right)^k\right)^{|S_2|}$.

[4]Some readers may note that a membership query of y into a Bloom filter $BF(S)$ is akin to creating a new filter from y, $BF(y)$, and determining whether $BF(S) \cap BF(y)$ is empty. It is straightforward to show that Theorem 2 represents this idea, as Eq. (8) reduces to the false positive probability of Lemma 1 for $|S_2| = 1$.

We use Lemma 2 by Guo *et al.*,

$$\Pr\left[BF(S_1) \cap BF(S_2) = BF(S_1 \cap S_2) \mid h \in \mathcal{H}_m\right]$$
$$= (1 - 1/m)^{k^2 \times |S_1 - S_1 \cap S_2| \times |S_2 - S_1 \cap S_2|}$$

but assume that the sets are disjoint:

$$\Pr\left[BF(S_1) \cap BF(S_2) = \emptyset \mid S_1 \cap S_2 = \emptyset, h \in \mathcal{H}_m\right]$$
$$= (1 - 1/m)^{k^2 |S_1||S_2|}. \tag{11}$$

Substituting (11) into (10) shows (7).

Regarding Eq. (8), to avoid a false set-overlap, partitioned Bloom filters require at least one partition to be empty, with all m/k bits set to zero. Thus the negation of this statement, a false set-overlap, results when all k partitions are non-empty. Consider any one single partition: note that it operates identically to an unpartitioned Bloom filter with length m/k bits, but only a single hash function indexing the sub-array. Eq. (7) of this theorem therefore suggests that a single Bloom filter partition of length m/k and one hash function is non-empty with probability

$$1 - \left(1 - \frac{1}{m/k}\right)^{(1)^2 |S_1||S_2|}. \tag{12}$$

Looking at the entire partitioned Bloom filter, we assume that the "emptiness" of all k partitions is mutually independent. Using (12), the probability that all k partitions are non-empty is

$$\left(1 - \left(1 - \frac{k}{m}\right)^{|S_1||S_2|}\right)^k,$$

completing the proof of Eq. (8). □

4. ANALYTICAL COMPARISON OF QUERYING AND INTERSECTION

In this section we analytically compare the statistical properties and space requirements of Bloom filter intersection and querying when determining address-set disjointness. Specifically, we demonstrate (i) that partitioned Bloom filters always outperform unpartitioned Bloom filters when determining address-set disjointness by intersection; (ii) that for equivalent probability of false set-overlap (PFSO), partitioned Bloom filter intersection requires a factor $\Omega(\sqrt{|S_2|})$ more space than performing a queue-of-queries (of set S_2) into a Bloom filter.

4.1 Preliminary Inequalities

We state elementary inequalities from Mitrinović *et al.* [22, 23] used to prove the main results of the section.

Lemma 3. [23] *Bernoulli's Inequality.* *If* $-1 < x < \frac{1}{n-1}, x \neq 0$, *and integer* $n = 2, 3, \ldots$, *then*

$$1 + nx < (1+x)^n < 1 + \frac{nx}{1 + (1-n)x}.$$

Lemma 4. [22] *Generalization of Bernoulli's Inequality.* *If* $0 < q < p$ *and* $-q < x < 0$, *then*

$$\left(1 + \frac{x}{q}\right)^q \geq \left(1 + \frac{x}{p}\right)^p.$$

Lemma 5. *If real* x *is such that* $0 < x < 1$ *and integer* $n > 1$, *then* $1 - x^n > 1 - x > (1-x)^n$.

Proof. $x \in (0,1) \Rightarrow x^n < x$, so evidently $1 - x^n > 1 - x$. Also, $(1-x) \in (0,1) \Rightarrow 1 - x > (1-x)^n$. □

4.2 Statistical Comparison of Bit-Indexing for Bloom Filter Intersection

The following theorem asserts that partitioned Bloom filter intersection has a lower PFSO than unpartitioned. It concerns two disjoint sets $S_1, S_2 \subset U$, that are represented by Bloom filters of the same length m, with the same hash function tuple. The k hash values are truly random, and we consider $m > k > 1$, since for a single hash function, partitioned Bloom filters are effectively unpartitioned.

Theorem 3. *Concerning false set-overlap by Bloom filter intersection, the partitioned bit-indexing scheme follows a probability distribution that is strictly less than that of an unpartitioned Bloom filter. That is,* $(\forall h_{m/k} \in \mathcal{H}_{\frac{m}{k}})(\forall h_m \in \mathcal{H}_m)$,

$$\Pr\left[\mathrm{FSO}_\cap(S_1, S_2, h_{m/k})\right] < \Pr\left[\mathrm{FSO}_\cap(S_1, S_2, h_m)\right].$$

Proof. We begin by using Lemma 4, substituting $x = -1/m$, $q = 1/k$, and $p = 1$, which satisfies $0 < q < p$ and $-q < x < 0$, to see that $\left(1 - \frac{k}{m}\right)^{\frac{1}{k}} \geq \left(1 - \frac{1}{m}\right)$. Additionally, since $m > k > 1$, then by Lemma 5, $\left(1 - \frac{1}{m}\right) > \left(1 - \frac{1}{m}\right)^k$. Combining these observations,

$$\left(1 - \frac{k}{m}\right)^{\frac{1}{k}} > \left(1 - \frac{1}{m}\right)^k$$
$$\Rightarrow \left(1 - \frac{k}{m}\right)^{|S_1||S_2|} > \left(1 - \frac{1}{m}\right)^{k^2 |S_1||S_2|}, \tag{13}$$

where the implication follows since $m > k > 1$, and we raise each side to the power of $(k|S_1||S_2|) > 0$. Rearranging Eq. (13), we show the main result,

$$\Pr\left[\mathrm{FSO}_\cap(S_1, S_2, h_m)\right] = 1 - \left(1 - \frac{1}{m}\right)^{k^2 |S_1||S_2|}$$
$$> 1 - \left(1 - \frac{k}{m}\right)^{|S_1||S_2|} \tag{14}$$
$$> \left(1 - \left(1 - \frac{k}{m}\right)^{|S_1||S_2|}\right)^k \tag{15}$$
$$= \Pr\left[\mathrm{FSO}_\cap(S_1, S_2, h_{m/k})\right], \tag{16}$$

where Eq. (15) follows from Lemma 5. □

4.3 Space Comparison of Intersection and Queue-of-Queries

Given that partitioned intersection outperforms unpartitioned intersection, the following theorem thus compares the methods queue-of-queries and partitioned Bloom filter intersection for deciding set disjointness. The metric of consideration is more concrete than that in the previous theorem: we will show that the bit-array space savings of queue-of-queries is at least a factor of the square root of set cardinality, relative to partitioned Bloom filter intersection, when the respective PFSOs are equal, under reasonable conditions.

Consider the disjoint sets $S_1, S_2 \subset U$. Let $BF_q(S_1)$ be a partitioned Bloom filter of length m_q bits with a truly random k-tuple hash function $h_q \in \mathcal{H}_{\frac{m_q}{k}}$, for use in a queue-of-queries. Let partitioned Bloom filters $BF_i(S_1), BF_i(S_2)$ have length m_i bits and be indexed by the truly random k-tuple hash function $h_i \in \mathcal{H}_{\frac{m_i}{k}}$, for use in Bloom filter intersection. Assume more than one hash tuple value, $k > 1$, nontrivial sets, $|S_1|, |S_2| > 1$, and assume bit-array lengths $m_q > k$ and $m_i > d|S_1||S_2| > k$, for some constant $d \geq 1$.

Theorem 4. *Assuming the preceding system and conditions, the bit-array space requirement of partitioned Bloom filter intersection is a factor $\Omega(\sqrt{|S_2|})$ larger than the queue-of-queries method, for equivalent PFSO. Specifically, if $k > 1$ and*

$$\Pr[\text{FSO}_\in(S_1, S_2, h_q)] = \Pr[\text{FSO}_\cap(S_1, S_2, h_i)],$$

then

$$m_i > m_q \frac{|S_2|^{\left(1-\frac{1}{k}\right)}}{1+\frac{k}{d}}. \tag{17}$$

Proof. Using the theorems of Section 3.2, we first equate the stated probabilities, then using the lemmas of Section 4.1, show the inequality between m_i and m_q. The following equality is given:

$$\left(1 - \left(1 - \frac{k}{m_i}\right)^{|S_1||S_2|}\right)^k$$
$$= 1 - \left(1 - \left(1 - \left(1 - \frac{k}{m_q}\right)^{|S_1|}\right)^k\right)^{|S_2|}.$$

For clarity, let $a = |S_1|$, $b = |S_2|$, and $c_i = \left(1 - \frac{k}{m_i}\right)^a$, and likewise for c_q. Applying these substitutions, we have

$$\left(1 - {c_i}^b\right)^k = 1 - \left(1 - (1 - c_q)^k\right)^b$$

which is rearranged into

$$1 - \left(1 - {c_i}^b\right)^k = \left(1 - (1 - c_q)^k\right)^b. \tag{18}$$

Observe that $c_i, c_q \in (0, 1)$ since $m_i, m_q > k$ and $a > 1$. Therefore $(1 - c_q)^k \in (0, 1)$, so with integer $b > 1$, we may apply the left side of Bernoulli's inequality (Lemma 3) to the right hand side of Eq. (18) and have

$$1 - \left(1 - {c_i}^b\right)^k = \left(1 - (1 - c_q)^k\right)^b > 1 - b\,(1 - c_q)^k.$$

Rearranging and simplifying terms,

$$b\,(1 - c_q)^k > \left(1 - {c_i}^b\right)^k.$$

Isolate b on the left hand side, take the k'th root, then expand c_i and c_q:

$$b^{\frac{1}{k}} > \frac{1 - {c_i}^b}{1 - c_q}$$
$$= \frac{1 - \left(1 - \frac{k}{m_i}\right)^{ab}}{1 - \left(1 - \frac{k}{m_q}\right)^{a}}. \tag{19}$$

These steps are valid as $c_i, c_q \in (0, 1)$. Now consider the denominator of (19). Apply the left side of Bernoulli's inequality (Lemma 3), since integer $a > 1$, and $m_q > k$. Then

$$1 - \left(1 - \frac{k}{m_q}\right)^a < \frac{ak}{m_q} \Rightarrow \frac{1}{\left(1 - \left(1 - \frac{k}{m_q}\right)^a\right)} > \frac{m_q}{ak}. \tag{20}$$

Now focus on the numerator of (19), $1 - \left(1 - \frac{k}{m_i}\right)^{ab}$. Using the right side of Lemma 3, we let $x = -\frac{k}{m_i}$, and $n = ab$, which satisfies $-1 < x < \frac{1}{n-1}$ since $m_i > k$. Thus,

$$\left(1 - \frac{k}{m_i}\right)^{ab} < 1 + \frac{ab\left(-\frac{k}{m_i}\right)}{1 + (1 - ab)\left(-\frac{k}{m_i}\right)}$$

Rearranging,
$$1 - \left(1 - \frac{k}{m_i}\right)^{ab} > \frac{\frac{kab}{m_i}}{1 + \frac{kab}{m_i} - \frac{k}{m_i}}$$
$$> \frac{\frac{kab}{m_i}}{1 + \frac{kab}{m_i}}$$
$$= \frac{kab}{m_i + kab} \tag{21}$$

Combining inequalities (20) and (21) into (19), we have

$$b^{\frac{1}{k}} > \frac{1 - \left(1 - \frac{k}{m_i}\right)^{ab}}{1 - \left(1 - \frac{k}{m_q}\right)^{a}} > \frac{\frac{kab}{m_i+kab}}{\frac{ka}{m_q}}$$
$$= b\frac{m_q}{m_i + kab}$$

Rearranging, we have shown thus far that

$$m_i + abk > m_q b^{1-\frac{1}{k}}.$$

For some constant $d \geq 1$, if designers choose $m_i > abd$, then $m_i\frac{k}{d} > abk$, thus $\left(1 + \frac{k}{d}\right)m_i > m_i + abk$. Therefore, returning $b = |S_2|$,

$$m_i > m_q \frac{|S_2|^{1-\frac{1}{k}}}{1+\frac{k}{d}},$$

as desired.

Regarding asymptotic notation for the space comparison, we claim that, $\forall k \geq 2$, $\frac{m_i}{m_q} = \Omega(\sqrt{|S_2|})$, or formally, $\forall k \geq 2$,

$$(\exists c, n_0 > 0)(\forall |S_2| \geq n_0) \quad \frac{m_i}{m_q} \geq c\sqrt{|S_2|}. \tag{22}$$

Here we show only the base case $k = 2$—the full proof can be found in Jeffrey's thesis [14].

For $k = 2$ and any $d \geq 1$, we have from Eq. (17),

$$\frac{m_i}{m_q} > \frac{|S_2|^{1-\frac{1}{k}}}{1+\frac{k}{d}} = \frac{\sqrt{|S_2|}}{1+\frac{2}{d}}$$

which satisfies (22) for $c = \frac{1}{1+\frac{2}{d}}$ and $n_0 > 0$. $\square$

5. EMPIRICAL VALIDATION

In this section we empirically validate the probability distributions derived in Section 3.2. Empirical rates of false set-overlap are gathered for each of the three address-set disambiguation methods, in four discrete Bloom filter configurations. A simple experiment tests two disjoint address-sets for overlap, using the three methods discussed: queue-of-queries, and partitioned and unpartitioned Bloom filter

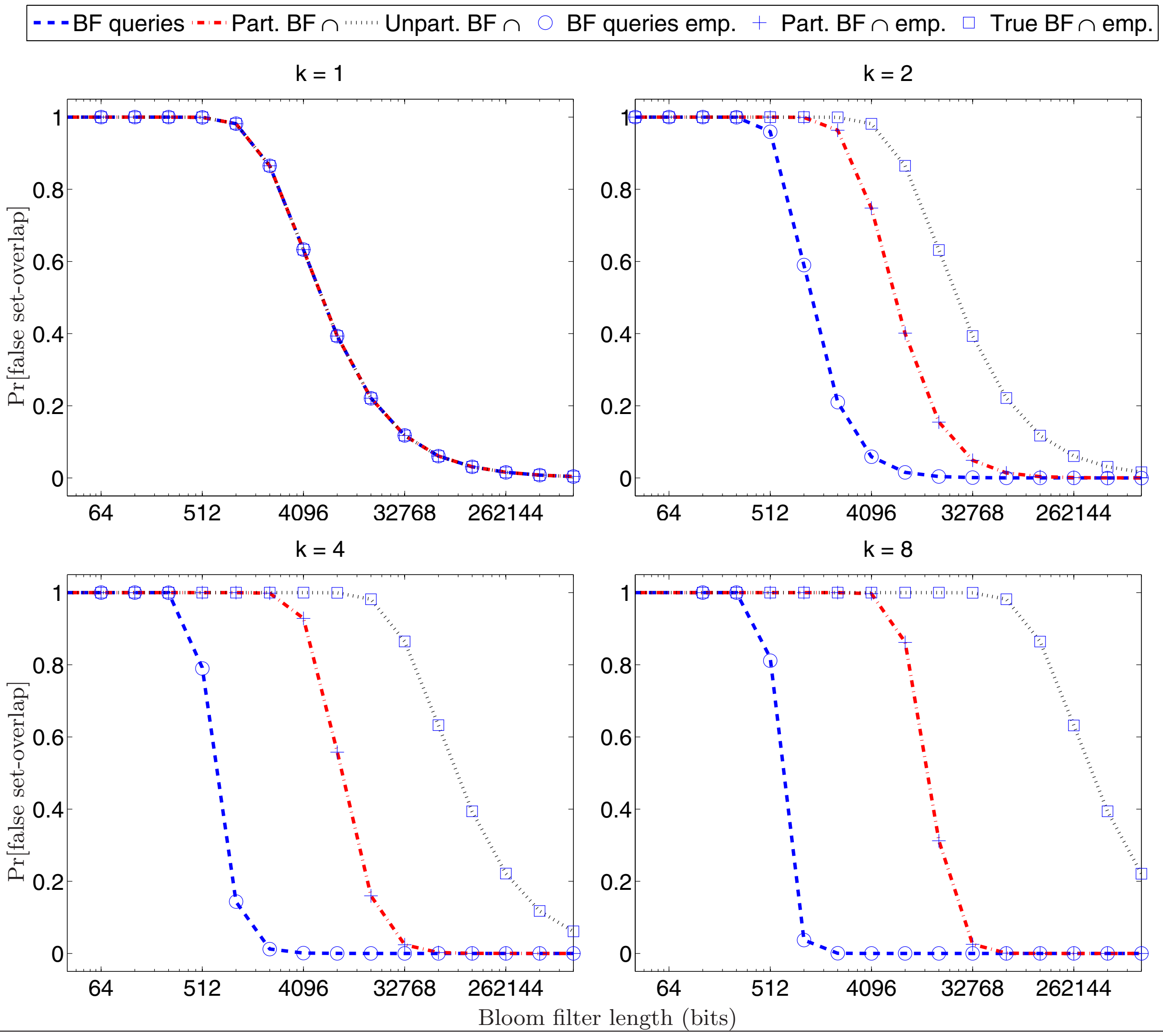

Figure 3: Probability and empirical rate of false set-overlap on the y-axis as they vary with increasing Bloom filter length on the x-axis (in log scale). Each plot represents a size k of the hash function tuple. The three curves plot the probabilistic models of false set-overlap, and are overlaid by sample points of the experimentally measured rate.

intersection. Each method returns two possible outcomes: either a false set-overlap, or disjoint sets. The experiment is repeated over 10^6 trials, and the relative frequency of false set-overlap is recorded as the empirical rate.

For a single experiment, two disjoint sets are generated, S_1 and S_2, containing random unique 32-bit integers (addresses), using the `C` standard library `rand` function. For a given bit-array length, m, and hash function tuple size, k, partitioned and unpartitioned Bloom filters are constructed for each of the sets, $BF_p(S_1), BF_p(S_2), BF_u(S_1), BF_u(S_2)$. Random hash functions are selected from the H_3 family [4] that approximately match the performance of ideal hash functions, for a sufficiently-random address stream [24]. To test the queue-of-queries outcome, each $x \in S_2$ is tested for membership in $BF_p(S_1)$. If at least one query returns true, the false set-overlap is recorded. Likewise, to test the Bloom filter intersection outcome, we separately intersect $BF_p(S_1) \cap BF_p(S_2)$ and $BF_u(S_1) \cap BF_u(S_2)$, and determine whether the remaining bits represent an empty set as described in Section 3.1; otherwise a false set-overlap is recorded.

Figure 3 visualizes the false set-overlap rate as a function of Bloom filter length, m. The four plots differ only in the size of the hash function tuple: $k = 1, 2, 4, 8$. The cardinalities of the address sets were fixed to $|S_1| = |S_2| = 64$ unique elements[5] for all trials. Each set of 10^6 trials is represented by a point on the plot, having been assigned a fixed filter length. Filter lengths are shown in a log scale on the x-axis, and each length was sampled at a power-of-two to effectively show the trend of this roughly exponential decay.

[5]Address set cardinality is certainly application-specific; a number of earlier studies displayed average read set sizes of 26 to 67 addresses [5, 6, 32, 33, 42], but as few as 2 addresses [18, 41] and as many as 2000 addresses [32, 42]. Write set sizes are typically smaller but still vary from 1 address [5, 17, 33, 41, 42] to over 1500 [32, 42]—we choose 64-element sets as a compromise within this large space. Varying the set cardinalities will not change the general trends observed in Figure 3.

The derived theoretical distributions underlay the empirical sample points, visualizing the relationship between Bloom filter length and false set-overlap probability. It is apparent that the empirical sample points follow the theoretical distributions, validating the accuracy of our work.

6. IMPLICATIONS

Figure 3 illustrates that varying k has a different effect on each method of deciding set disjointness. For nontrivial hash function tuples (i.e., $k > 1$), the queue-of-queries method benefits from increasing k (up to a point), while the probability distribution for unpartitioned Bloom filter intersection only becomes worse. Upon close inspection of the distribution for partitioned Bloom filters, increasing k is beneficial only for filter lengths of at least 16kbits. These three patterns can also be shown analytically, by minimizing the probability distributions with respect to k. Due to space constraints, the reader is directed to Broder and Mitzenmacher [3] for an example of this process. Without proof, the optimal number of hash functions for partitioned intersection is $k^* = \frac{m}{|S_1||S_2|} \ln 2$, minimizing the false conflict probability to 2^{-k^*}.

Implication 1. *Issuing a series of Bloom filter membership queries provides set disambiguation with significantly lower space overhead than Bloom filter intersection, for larger than one-tuple hash functions.*

Implication 2. *When intersection is unavoidable for deciding pairwise set disjointness, partitioned intersection is preferable to unpartitioned as it provides the same service, with the same time-complexity, with lower (or equal) probability of false set-overlap.*

Surprisingly, for a single hash function, $k = 1$, all three methods share the same probability distribution (and empirical sample points)—this can be verified by substituting $k = 1$ for the theorems of Section 3, and the PFSO is $1 - (1 - \frac{1}{m})^{|S_1||S_2|}$. Given this equivalence, a time-complexity comparison would be helpful; unfortunately the many hardware-dependent considerations make such a study beyond the scope of this paper.

Implication 3. *Remarkably, when restricted to encoding addresses using a single hash function, Bloom filter querying and intersection share equivalent probability of false set-overlap.*

7. CONCLUSION

Motivated by the recently-popular use of Bloom filters for lazy address-set disambiguation, in this paper we introduced and conclusively compared probabilistic models for the three methods of using Bloom filters to decide set disjointness: (i) queue-of-queries, and intersection of (ii) unpartitioned and (iii) partitioned Bloom filters. We analytically and graphically demonstrated that the intersection of partitioned Bloom filters has more desirable probability of false set-overlap than their unpartitioned counterparts. We also demonstrated that partitioned intersection requires at least a factor of the square root of set cardinality more bit-array space than a queue-of-queries approach, to maintain the same probability of false set-overlap. Finally, we observed that when designers are (unfortunately) required to use a one-tuple hash function, the queue-of-queries and intersection methods share identical probability of false set-overlap; they should use the most time-efficient strategy in such a case. The Bloom filter is indeed an excellent fit to address-set disambiguation for parallelization systems and tools, but this increasingly-common yet unconventional use for deciding set-overlap demands more study. We provide system designers with new insight in this area, easing Bloom filter design space exploration.

Acknowledgments

We thank the anonymous reviewers for their detailed comments. Bruce Francis was of monumental help through his tutorial, *Elements of Mathematical Style*, and his general feedback on our analytical work. We thank Hratch Mangassarian for the discussion on asymptotic notation, and James Tuck for initial discussion on the need for better theoretical understanding of Bloom filters in address-set disambiguation. Mark Jeffrey was supported by the NSERC Alexander Graham Bell Canada Graduate Scholarship (CGS-M).

8. REFERENCES

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

[2] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of Bloom filters. *Inf. Process. Lett.*, 108(4):210–213, 2008.

[3] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1:485–509, January 2004.

[4] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[5] L. Ceze, J. Tuck, P. Montesinos, and J. Torrellas. Bulksc: bulk enforcement of sequential consistency. *SIGARCH Comp. Arch. News*, 35(2):278–289, 2007.

[6] L. Ceze, J. Tuck, J. Torrellas, and C. Cascaval. Bulk disambiguation of speculative threads in multiprocessors. In *International Symposium on Computer Architecture*, 2006.

[7] K. Christensen, A. Roginsky, and M. Jimeno. A new analysis of the false positive rate of a Bloom filter. *Inf. Processing Letters*, 110(21):944 – 949, 2010.

[8] J. E. Gottschlich, M. Vachharajani, and J. G. Siek. An efficient software transactional memory using commit-time invalidation. In *International Symposium on Code Generation and Optimization*, 2010.

[9] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo. The dynamic Bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, 22:120–133, 2010.

[10] L. Hammond, M. Willey, and K. Olukotun. Data speculation support for a chip multiprocessor. In *Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[11] L. Han, W. Liu, and J. M. Tuck. Speculative parallelization of partial reduction variables. In *International Symposium on Code Generation and Optimization*, 2010.

[12] M. Herlihy and J. E. B. Moss. Transactional memory: architectural support for lock-free data structures. In *Intl. Symposium on Computer Architecture*, 1993.

[13] D. R. Hower and M. D. Hill. Rerun: Exploiting episodes for lightweight memory race recording. In *Intl. Symposium on Computer Architecture*, 2008.

[14] M. Jeffrey. Modeling Bloom filter intersection for address-set disambiguation. Master's thesis, University of Toronto, June 2011.

[15] V. Krishnan and J. Torrellas. A chip multiprocessor architecture with speculative multithreading. *IEEE Transactions on Computers, Special Issue on Multithreaded Architecture*, September 1999.

[16] M. Labrecque, M. Jeffrey, and J. G. Steffan. Application-specific signatures for transactional memory in soft processors. In *Intl. Symposium on Applied Reconfigurable Computing*, 2010.

[17] B. Lucia, L. Ceze, and K. Strauss. Colorsafe: architectural support for debugging and dynamically avoiding multi-variable atomicity violations. *SIGARCH Comput. Archit. News*, 38(3):222–233, 2010.

[18] B. Lucia, J. Devietti, L. Ceze, and K. Strauss. Atom-aid: Detecting and surviving atomicity violations. *IEEE Micro*, 29(1):73 –83, Jan.-Feb. 2009.

[19] M. Mehrara, J. Hao, P.-C. Hsu, and S. Mahlke. Parallelizing sequential applications on commodity hardware using a low-cost software transactional memory. In *Conference on Programming Language Design and Implementation*, 2009.

[20] L. Michael, W. Nejdl, O. Papapetrou, and W. Siberski. Improving distributed join efficiency with extended Bloom filter operations. In *International Conference on Advanced Networking and Applications*, 2007.

[21] C. C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun. An effective hybrid transactional memory system with strong isolation guarantees. In *Intl. Symposium on Computer Architecture*, 2007.

[22] D. S. Mitrinović and J. E. Pečarić. Bernoulli's inequality. *Rendiconti del Circolo Matematico di Palermo*, 42:317–337, 1993.

[23] D. S. Mitrinović and P. M. Vasić. *Analytic Inequalities*. Springer-Verlag, Berlin, 1970.

[24] M. Mitzenmacher and S. Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA '08: ACM-SIAM Symposium On Discrete Algorithms*, 2008.

[25] P. Montesinos, L. Ceze, and J. Torrellas. Delorean: Recording and deterministically replaying shared-memory multiprocessor execution efficiently. In *Intl. Symposium on Computer Architecture*, 2008.

[26] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood. Logtm: log-based transactional memory. In *International Symposium on High-Performance Computer Architecture*, 2006.

[27] J. K. Mullin. Estimating the size of a relational join. *Information Systems*, 18(3):189 – 196, 1993.

[28] A. Muzahid, D. Suárez, S. Qi, and J. Torrellas. Sigrace: signature-based data race detection. *SIGARCH Comp. Arch. News*, 37(3):337–348, 2009.

[29] O. Papapetrou, W. Siberski, and W. Nejdl. Cardinality estimation and dynamic length adaptation for Bloom filters. *Distributed and Parallel Databases*, 28:119–156, 2010.

[30] L. Peng, L. guo Xie, X. qiang Zhang, and X. yan Xie. Conflict detection via adaptive signature for software transactional memory. In *International Conference on Computer Engineering and Technology*, 2010.

[31] G. Pokam, C. Pereira, K. Danne, R. Kassa, and A.-R. Adl-Tabatabai. Architecting a chunk-based memory race recorder in modern cmps. In *International Symposium on Microarchitecture*, 2009.

[32] R. Quislant, E. Gutierrez, O. Plata, and E. L. Zapata. Improving signatures by locality exploitation for transactional memory. In *Intl. Conference on Parallel Architectures and Compilation Techniques*, 2009.

[33] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam. Implementing signatures for transactional memory. In *International Symposium on Microarchitecture*, 2007.

[34] A. Shriraman, S. Dwarkadas, and M. L. Scott. Flexible decoupled transactional memory support. In *Intl. Symposium on Computer Architecture*, 2008.

[35] A. Shriraman, S. Dwarkadas, and M. L. Scott. Implementation tradeoffs in the design of flexible transactional memory support. *J. Parallel Distrib. Comput.*, 70, October 2010.

[36] M. F. Spear, M. M. Michael, and C. von Praun. Ringstm: scalable transactions with a single atomic instruction. In *Symposium on Parallelism in Algorithms and Architectures*, 2008.

[37] J. G. Steffan and T. C. Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In *International Symposium on High-Performance Computer Architecture*, pages 2–13, 1998.

[38] J. Torrellas, L. Ceze, J. Tuck, C. Cascaval, P. Montesinos, W. Ahn, and M. Prvulovic. The bulk multicore architecture for improved programmability. *Commun. ACM*, 52(12):58–65, 2009.

[39] M. Waliullah and P. Stenstrom. Efficient management of speculative data in hardware transactional memory systems. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2008.

[40] S. Wang, D. Wu, Z. Pang, and X. Yang. Software assisted transact cache to support efficient unbounded transactional memory. In *Intl. Conference on High Performance Computing and Communications*, 2008.

[41] L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood. Logtm-se: Decoupling hardware transactional memory from caches. In *International Symposium on High Performance Computer Architecture*, 2007.

[42] L. Yen, S. C. Draper, and M. D. Hill. Notary: Hardware techniques to enhance signatures. In *International Symposium on Microarchitecture*, 2008.

[43] P. Zhou, R. Teodorescu, and Y. Zhou. Hard: Hardware-assisted lockset-based race detection. In *International Symposium on High Performance Computer Architecture*, pages 121–132, 2007.