Minimizing Page Faults on Bloom Filters

Adam Zawierucha (adz2)

Rice University
zawie@rice.edu

Abstract

Bloom filters [1] are used ubiquitously due to their speed and memory efficiency in theory and in practice. However, the standard implementation of sufficiently large bloom filters suffers from page faults. In this paper we propose a bloom filter implementation that guarantees one page access per insert or query. This minimizes page faults, thereby drasticaly improving efficiency. We will show theoretically and empirically that our hierarchical implementation is expected to be faster than the standard implementation.
Keywords: probabilistic data structures, bloom filters, memory hierarchy, implementation, page fault analysis

1 Introduction

In this section I will describe the proposed hierarchial blooom filter implementaiton in detail.

Our hierarchial bloom filter will be a collection of bloom filters of the computer systems page size (4096 bytes). We will have as many bloom filters as it takes to occupy M bits. M sohuld be $10N$ where N is the number of expected keys to be in the bloom filter. We will generate 8 independent hash functions. 1 hash function will be used to select the sub-bloom filter; the other 7 will be used to set bits in the baby bloom filter as the standard implementation.

The idea behind this is that we minimize page faults, as we only need to access one page of memory to set bits instead of up to 7.

The numbers selected will be justified in the *Theoretical Justification* section.

2 Literature Survey

In my literature survey, I discovered a similar approach except making bloom filters hierarchial. Tim Kaler's proposed cache-efficient bloom filters [3] makes sub-bloom filters of size cache-size; thus, their idea is very similar to mine except they do it on a smaller unit of memory.

Evgeni Krimer and Mattan Erez used a power-of-two choice principle within blocked-bloom filters to decrease the false positive rate. Instead of simply selecting one block to write into, they choose multiple. [2].

3 Performance Theoretical Justification

4 Experiments

We will now emperically validate that the hierarchical implementation is more time efficient than the standard implementation without sacrificying accuracy. Three experiments will be conducted.

First, we will **measure elapsed time as insertions scale** of each implementation. We anticipate that the hierarchical implementation will take less time than the standard implementation for any sufficiently large value of insertions.

Second, we will **measure false positives as we scale bits allocated per element** of each implementation. We anticipate that the hierarchial and standard implementation will be approximately same as the theoerical false positive rate discussed before.

Third, we will **measure throughput as insertions scale with a fixed bitarray size.** The primary intention of this experiment is to explore the hierarchial implementation's efficiency for small bitarray

sizes. We do not anticipate the hierarchial implementation to outperform the standard implementation for bitarray's of less than one page size.

For each of these sections, we pseudorandomly generate keys to both insert and query. Discussion on exactly how these keys are generated is discussed in the appendix.

4.1 Comparing Time Efficiency

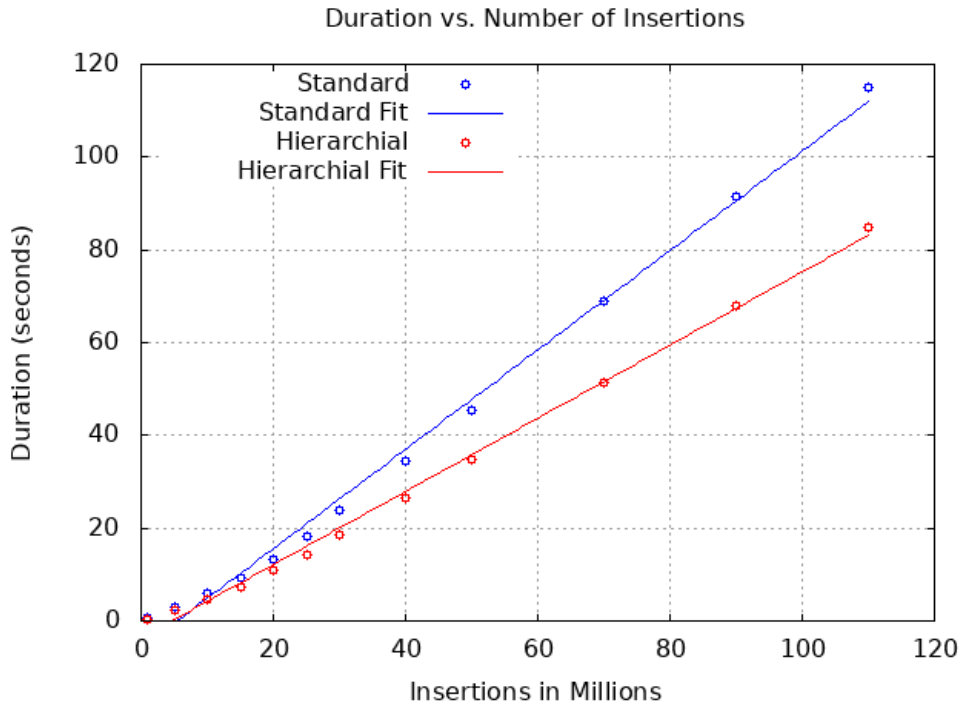
For this experiment, we seek to validate that the hierarchical implementation performs better than the standard implementation as the number of insertions grow. *Hypothesis:* The hierarchial bloom filter will take less time than the standard implementation to insert n keys for any n.

4.1.1 Experimental Settings

The experiment will run as follows:

1. Generate n random keys.
2. Generate a bloom filter of both varianets of size $10n$. Use the optimal theoeretical configuraiton for each bloom filter (i.e, $k = 7$, $l = 1$).
3. Time how long it takes to insert all n keys into each of the bloom filters.
4. Repeat for various sizes of n.

4.1.2 Results



The slope of the line of best fits that are plotted are as follows whre n is millions of insertions:

- The best fit line for the standard implementation is: $t = 1.07146 \cdot n - 5.83917$
- The best fit line for the hierarchical implementation is: $t = 0.789073 \cdot n - 3.64732$

Thus, our experiment shows that the hierarchical implementation performs 35% more operations per second than the standard implementation!

4.1.3 Conclusion

We have supported our hypothesis that the hierarchical implementation is more efficient than the standard implementation. In fact, it can perform 35% more operations per second.

References

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

[2] Mattan Erez Evgeni Krimer. The power of 1+alpha for memory efficient bloom filters. *Electrical and Computer Engineering Department University of Texas at Austin*. URL `http://lph.ece.utexas.edu/users/krimer/pub/im11_bf.pdf`.

[3] Time Kayler. Cache efficient bloom filters for shared memory machines. *MIT Computer Science and Artificial Intelligence Laboratory*. URL `http://tfk.mit.edu/pdf/bloom.pdf`.