Project Milestone Document

Title: Improving Co-operative Caching Using Importance Aware Bloom Filter

Milestone: 1

Advisor: Dr. Prof. Hans- Peter Bischof

Student: Shridhar Bhalekar (snb3300@rit.edu)

Problem Statement

Bloom Filter and its variants [2] are widely used data structure in distributed environments. One of the major deployment areas of Bloom Filters is collaborative caching in distributed environments. One of the real world examples of such system is multiple web proxies sharing their caches with other peers. To make sure such a distributed system works correctly these Bloom Filters must perform efficiently.

First part of this project is to compare the Importance Aware Bloom Filter with conventional Bloom Filter. Second part of this project is to compare cooperative caching algorithms. Summary Cache [1] will be modified to use Importance Aware Bloom Filter and it will then be compared with Greedy Forwarding, N-Chance and Robinhood algorithms.

Strategy

Simulation technique will be used for both the parts of this project. For the comparison of Bloom Filter and Importance Aware Bloom Filter [3], simulator will create two identical networks of clients and server. On one system clients and server will be using Bloom Filter while on the other system they will use Importance Aware Bloom Filter. Simulator will distribute data on both networks and fire queries to get the number of False Positives.

For the comparison of Summary Cache (Importance Aware Bloom Filter) with N-Chance, Greedy Forwarding and Robinhood the simulator will create identical networks for all the algorithms. Simulator will distribute data on all networks and simulate query forwarding. During query forwarding it will keep track of the network hops, cache reference and disk reference. For the comparison of these algorithms I will be using ticks (time needed to access disk, cache and network hops) and cache hit/miss ratio.

Progress

My first task was to read the research papers and understand all the algorithms. After that I decided on the comparison metrics for both the parts. Next was to decide the experiment types and how to prove or disprove the hypothesis and I decided to use simulation for. Then, I finalized Java as the language for implementation and designed the simulator and came up with the class hierarchy

needed to execute the experiments. Next part was to implement the simulator with all the necessary classes.

I have successfully completed the comparison of Bloom Filter and Importance Aware Bloom Filter. Following was the experiment execution strategy:

1. Simulator reads in the experiment parameters and creates two networks of clients and server accordingly.
2. Simulator reads in the data file and distribute data on both the systems identically. This means cache contents of clients on different network but with same id will be identical.
3. Simulator creates the specified number of queries from the same data file randomly.
4. Simulator fires these queries on both the systems and records the false positives occurred.
5. This experiment was repeated for different number of queries.

Bloom Filter vs. Importance Aware Bloom Filter Results:

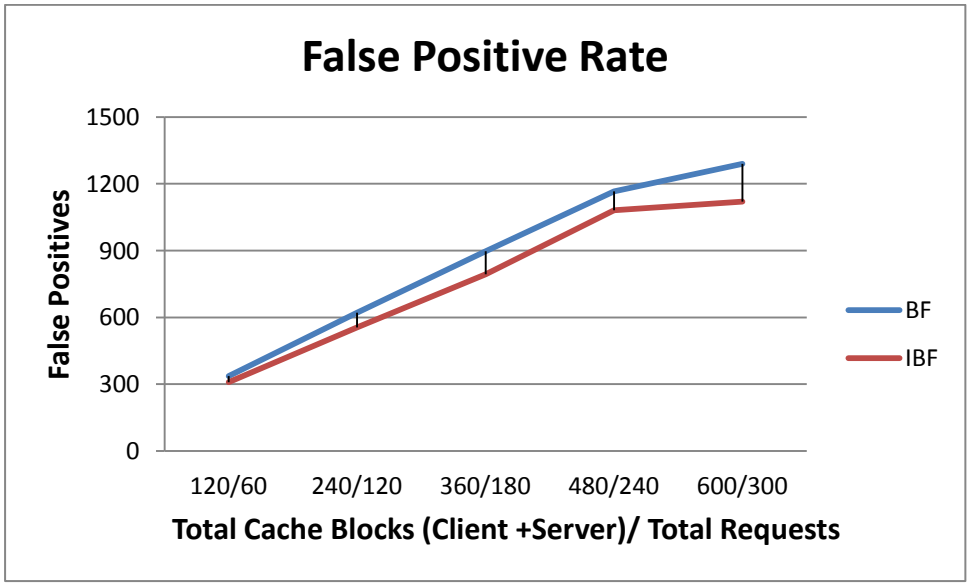
1. Comparing False Positive with respect to Cache Size and Total Requests

Experiment Overview

- Number of clients constant throughout the experiment.
- At each run change the cache size and number of requests
- Forward requests to clients
- Calculate False positives

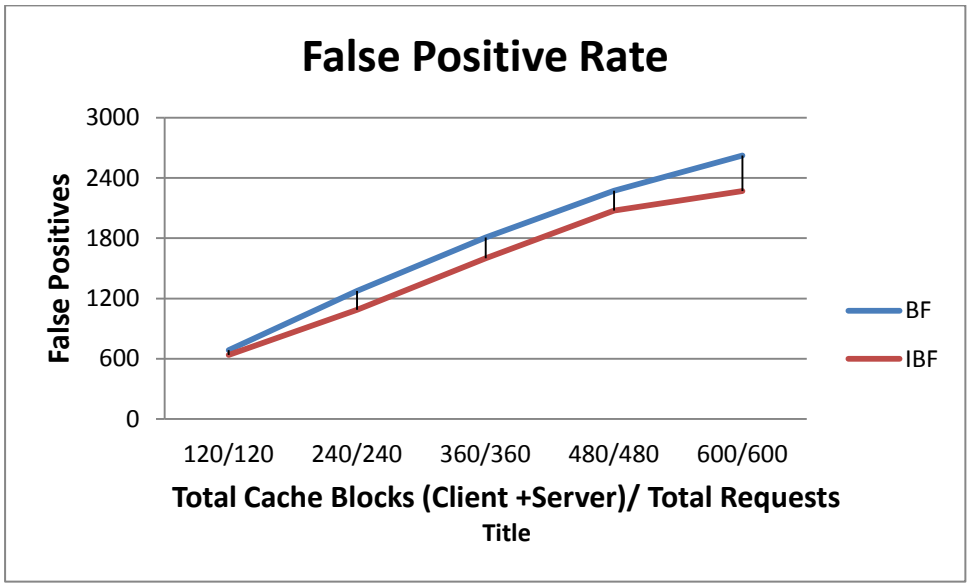
In the below 3 graphs x-axis shows the ratio of total cache blocks in the system and total requests forwarded and y-axis shows the false positives. Total cache blocks are the addition of total cache blocks in all clients and server.

1. Total Cache Blocks greater than total requests



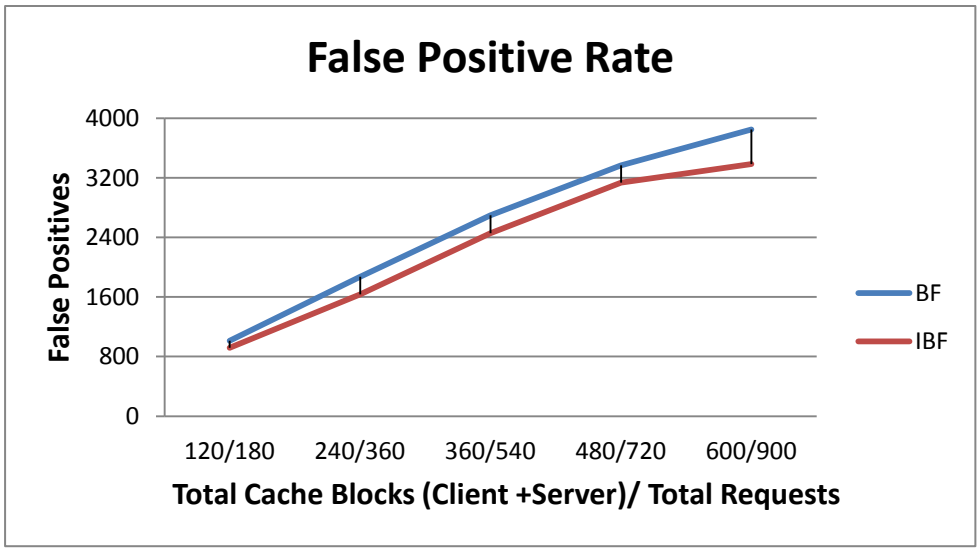
The above graph shows the scenario where cache size is greater than requested data. At each step the total requests are half the total cache blocks. The above graph shows that as the cache size increases Importance Aware Bloom Filter performs better than Bloom Filter although the false positive rate increases.

2. Total Cache Blocks equal to the total requests



The above graph shows the scenario where cache size is equal to requested data. At each step the total requests are equal to the total cache blocks. The above graph shows that as the cache size increases Importance Aware Bloom Filter performs better than Bloom Filter although the false positive rate increases.

3. Total Cache blocks less than total requests



The above graph shows the scenario where cache size is less than requested data. At each step the total requests are 1.5 times total cache blocks. The above graph shows that as the cache size increases Importance Aware Bloom Filter performs better than Bloom Filter although the false positive rate increases.

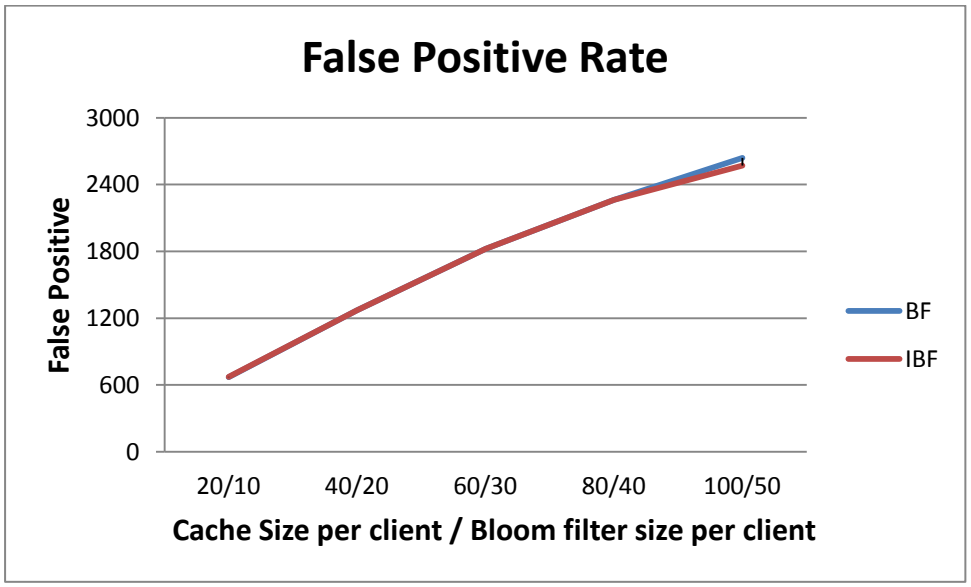
2. Comparing False Positive with respect to Cache Size and Bloom Filter Size

Experiment Overview

- Number of clients constant throughout the experiment.
- At each run keep the total requests equal to the total cache blocks of client and server combined
- At each run change the cache size and bloom filter size
- Forward requests to clients
- Calculate False positives

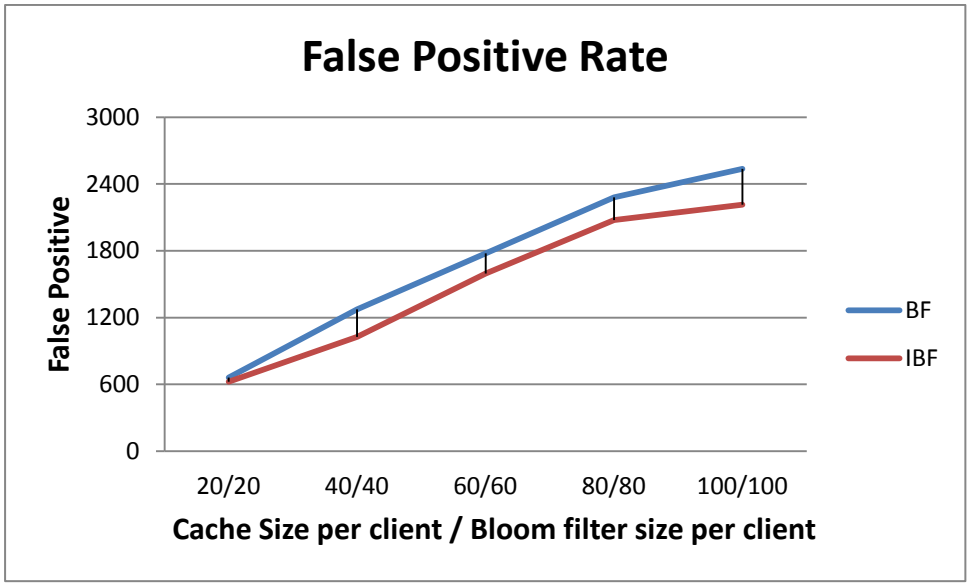
In the below 3 graphs x-axis shows the ratio of cache size per client and bloom filter size and y-axis shows the false positives. Total cache blocks are the addition of total cache blocks in all clients and server.

1. Bloom Filter size less than the total cache blocks per client/server



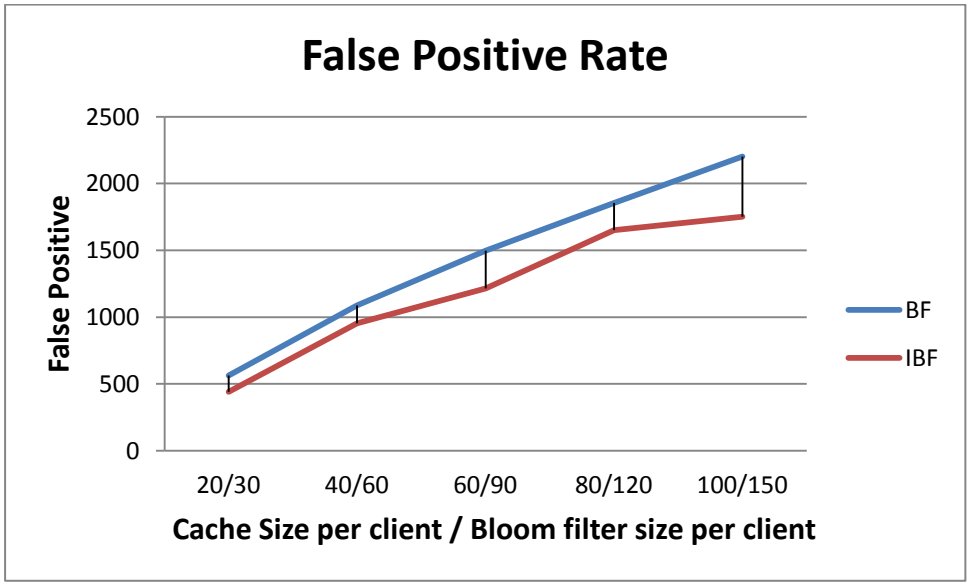
The above graph shows the scenario where bloom filter size is half the cache size. From the above graph it is evident that if bloom filter size is small then Importance aware bloom filter and bloom filter behave similarly and there is not enough difference in their performance.

2. Bloom Filter size equal to the total cache blocks per client/server



The above graph shows the scenario where bloom filter size is equal to the cache size. From the above graph it is evident that Importance Aware Bloom filter performs better than bloom filter. For small cache size their behaviour is similar but as cache size and bloom filter size increases Importance Aware Bloom filter performs better than Bloom Filter.

3. Bloom Filter size greater than total cache blocks per client/server



The above graph shows the scenario where bloom filter size is greater than cache size. From the above graph it is evident that Importance Aware Bloom filter performance is way better than bloom filter. As cache size increases performance of Bloom Filter deteriorates as compared to Importance Aware Bloom filter.

Future Plans

The next step of my project is to implement the remaining two algorithms i.e. Robinhood and Summary Cache with Importance Aware Bloom filter. This experiment will be a trace based simulation where in the random behaviour will be compared with four traces. These four traces are nothing but the best to worst test cases. Best case being the requested data is on the client cache while worst case being the requested data in on server disk.

References

[1] **Li Fan, P. C. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM* (pp. 281-293). IEEE/ACM.**

[2] Ming Zhong, P. L. (2008). Optimizing Data Popularity Concious bloom filters. *PODC '08 Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing.* ACM.

[3] Puru Kulkarni, R. B. (2013). Importance-aware Bloom Filter for Set Membership Queries in Streaming Data. *COMSNETS, 2013 Fifth International Conference.* COMSNETS.

[4] S. Tarkoma, C. E. (2012). Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Communications Surveys and Tutorials. Vol. 14, Number 1.* IEEE.