# 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, local/global cache) and cache hit/miss ratio.

### *Previous Milestones*

Following is the work I did in the previous milestones along with the analysis of the results which I got in milestone 1.

**Milestone 1**

In this milestone, I read the research papers and understood the algorithms and bloom filters. Then I decided the comparison metrics for bloom filters and cooperative caching algorithms. I used the following 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:*

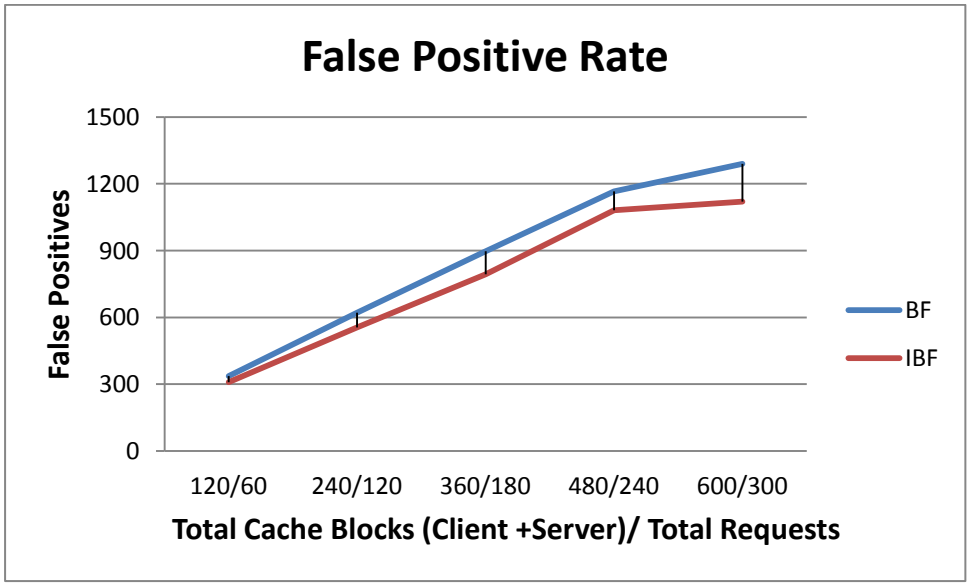
### 1. Comparing False Positive with respect to Cache Size and Total Requests

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

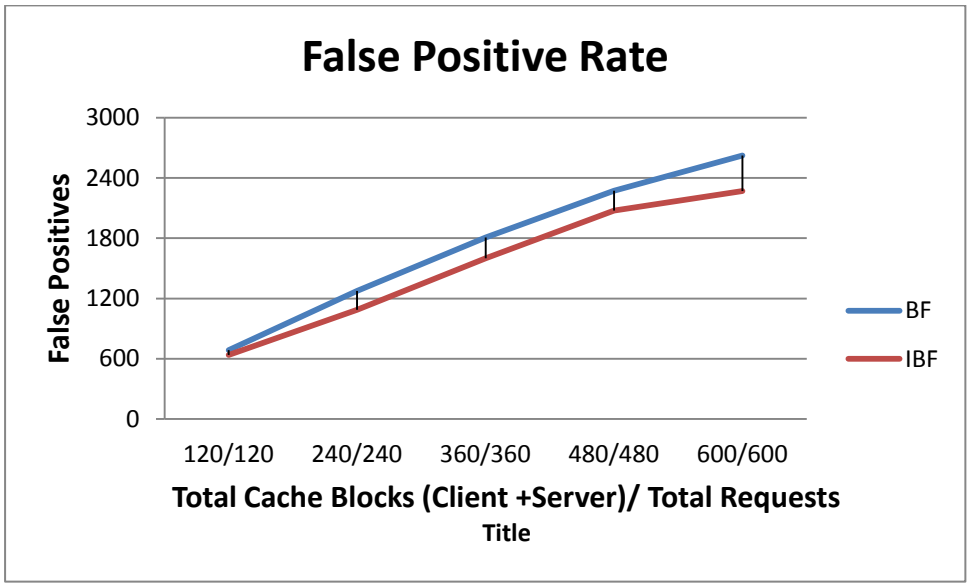
In the below 3 graphs x-axis shows the ratio of total cache blocks in the system and total requests forwarded and y-axis shows the false positives. Total cache blocks are the addition of total cache blocks in all clients and server.

1. Total Cache Blocks greater than total requests



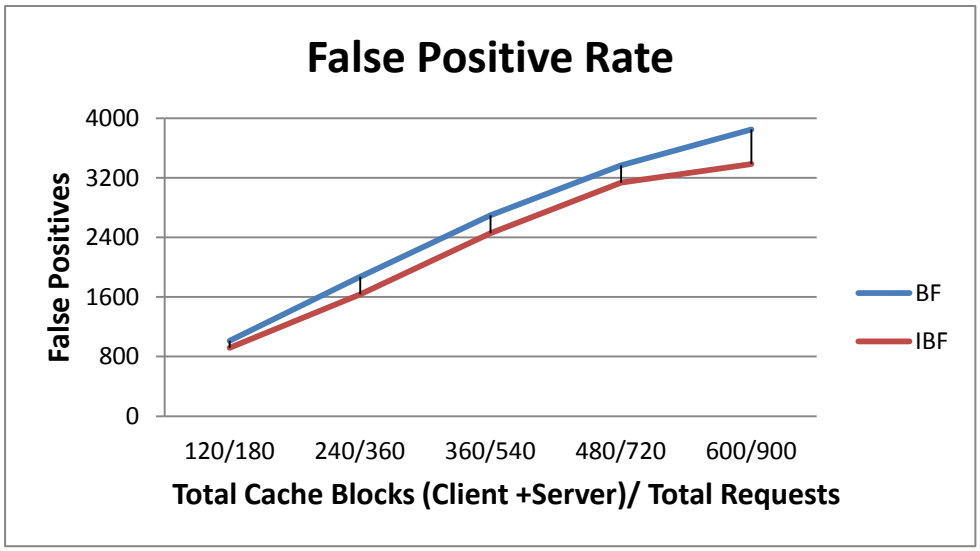
The above graph shows the scenario where cache size is greater than requested data. At each step the total requests are half the total cache blocks. The above graph shows that as the cache size increases Importance Aware Bloom Filter performs better than Bloom Filter although the false positive rate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

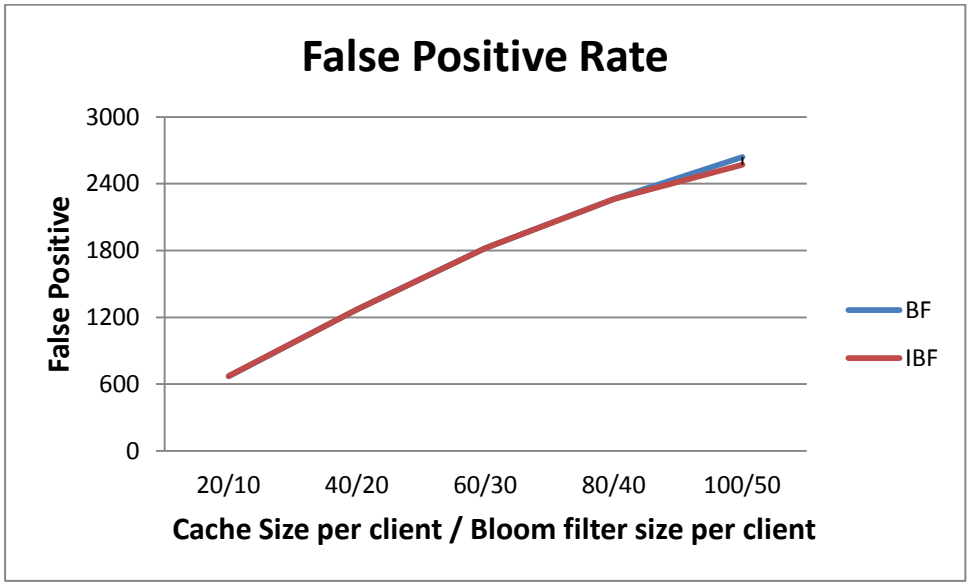
**2. Comparing False Positive with respect to Cache Size and Bloom Filter Size**

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

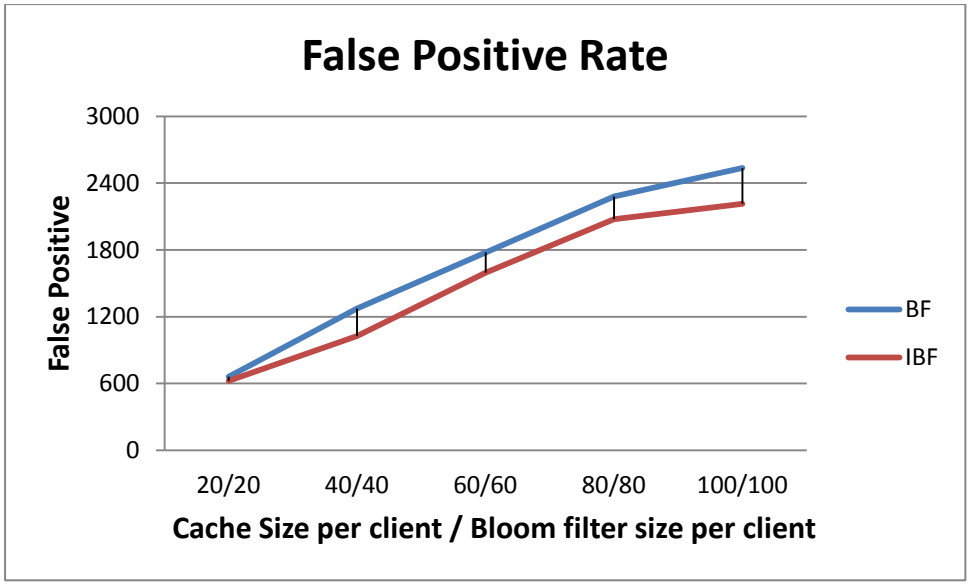
In the below 3 graphs x-axis shows the ratio of cache size per client and bloom filter size and y-axis shows the false positives. Total cache blocks are the addition of total cache blocks in all clients and server.

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



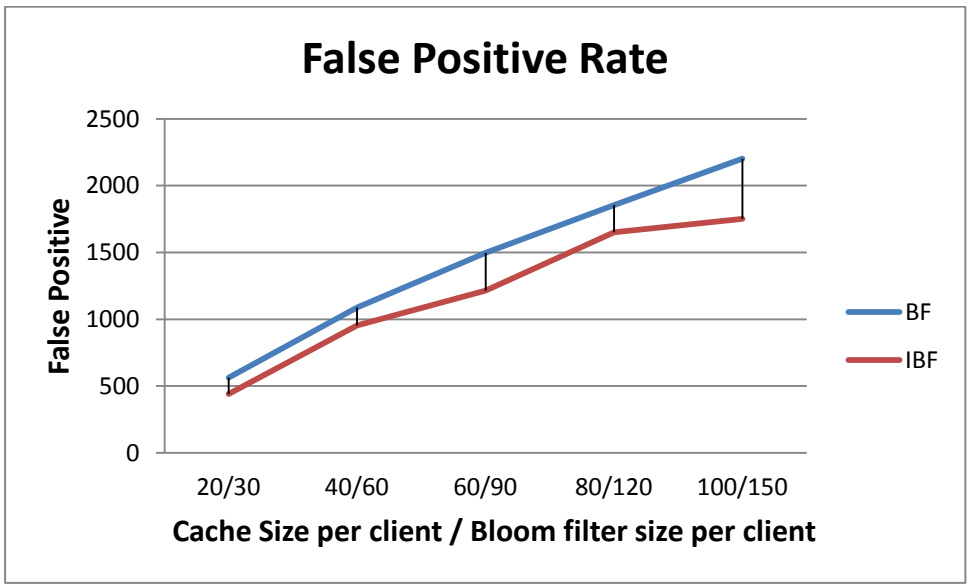

The above graph shows the scenario where bloom filter size is half the cache size. From the above graph it is evident that if bloom filter size is small then Importance aware bloom filter and bloom filter behave similarly and there is not enough difference in their performance.

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.

These experiments were executed for varying cache size, bloom filter size and total requests. From the results it was observed that Importance Aware Bloom Filter is better than Bloom Filter in terms of the False Positive rate as the cache size and bloom filter size increases.

**Milestone 2**

In this milestone I worked on the design and implementation of Greedy Forwarding, N-Chance and Robinhood algorithms. I modified the simulator code to support these algorithms. No results were obtained in this milestone. My entire code base is at https://github.com/snb3300/capstone/

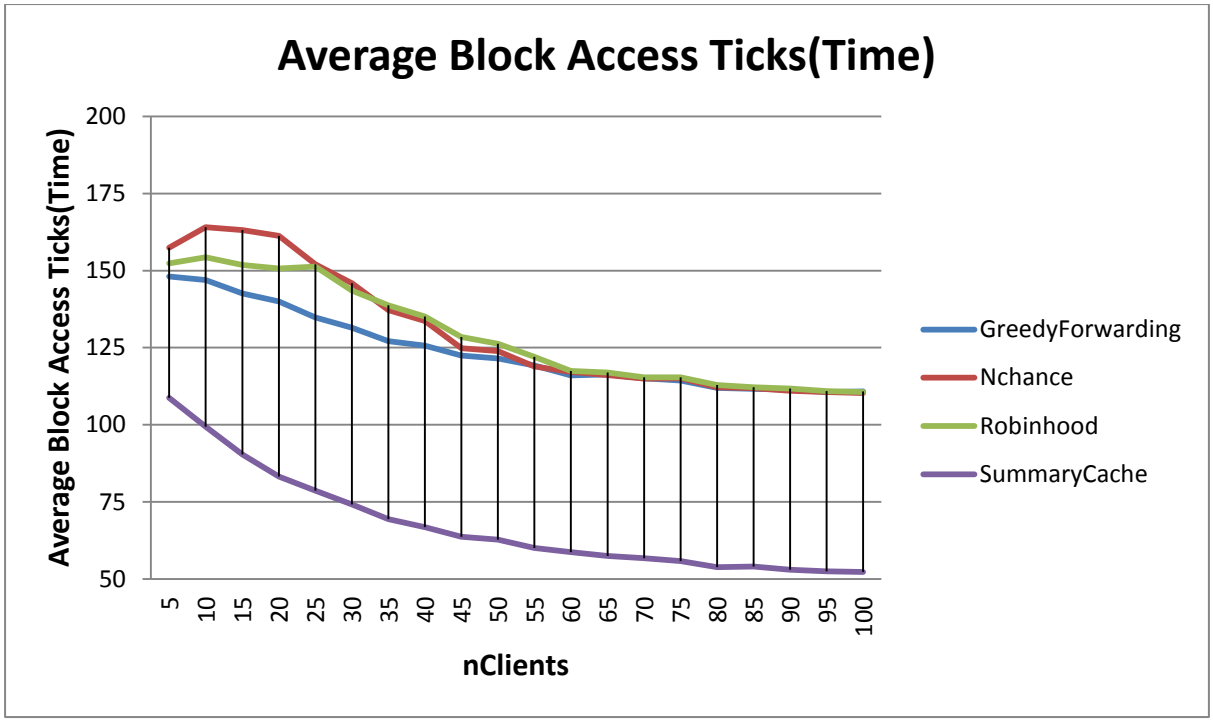
## *Progress*

This is the last milestone of my project. In this milestone I completed the implementation part of Robinhood and Summary Cache with Importance Aware Bloom Filter. After code completion I ran the experiments to collect the results. Following is the experiment execution strategy:

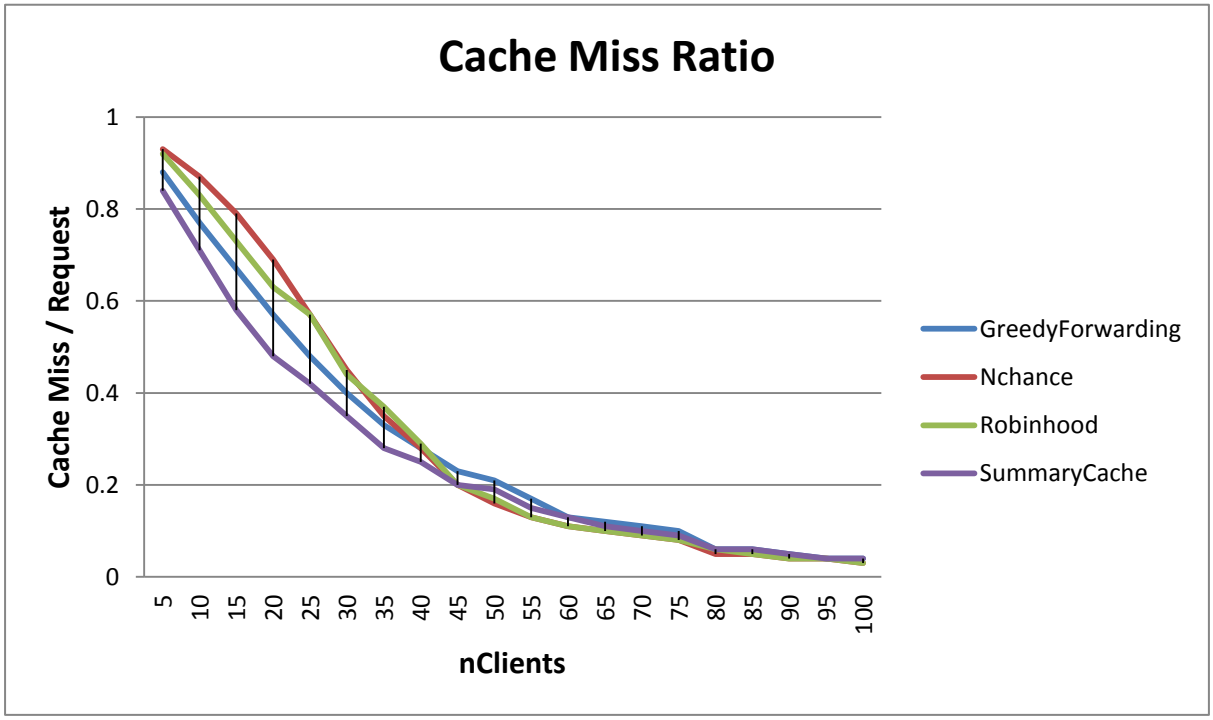
1. Simulator reads in the experiment parameters and creates a network of clients and server for all the four algorithms.
2. Simulator reads in the data file and distribute data on all the systems identically. This means cache contents of clients on different network but with same id will be identical.
3. Simulator creates the specified number of queries from the same data file randomly.
4. Simulator fires these queries on all the systems and records the average ticks required by each query and the cache hit/miss ratio.
5. The experiment is repeated with increasing number of clients.
6. In these simulation experiments I have assumed the average time required for local cache reference, global cache reference and disk reference. I have referred this tutorial to approximate the relative values I have used for different memory references.

Experiment parameters:

1. Total Disk Size = 1000 blocks (constant)
2. Client Cache Size = 20 blocks (constant)
3. Total Clients = 5 – 100 (variable)

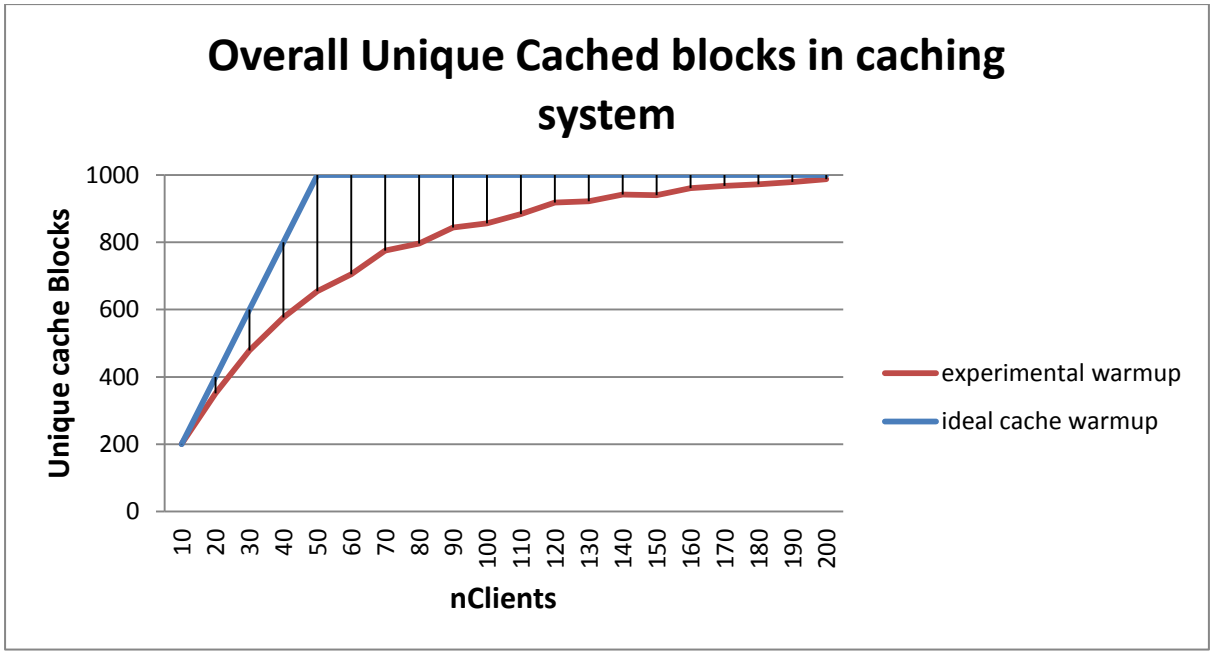



The above graph displays the average ticks (time) required to serve requests by the cooperative caching system. As the number of clients increase the total cache size increases and more and more requests are served from local or global cache. Due to this phenomenon the average block access ticks eventually decreases.

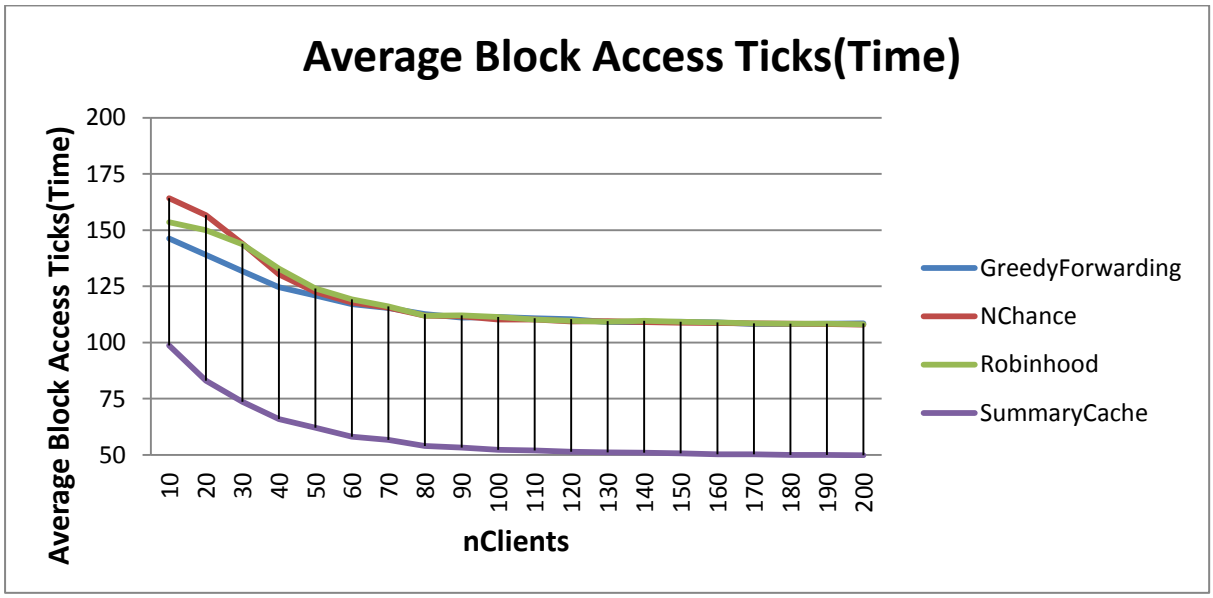



The above graph displays the cache miss ratio while serving requests by the cooperative caching system. This graph proves that the as the cache miss decreases more and more requests are served from local/global caches. As the total number of clients goes beyond 50 the total cache

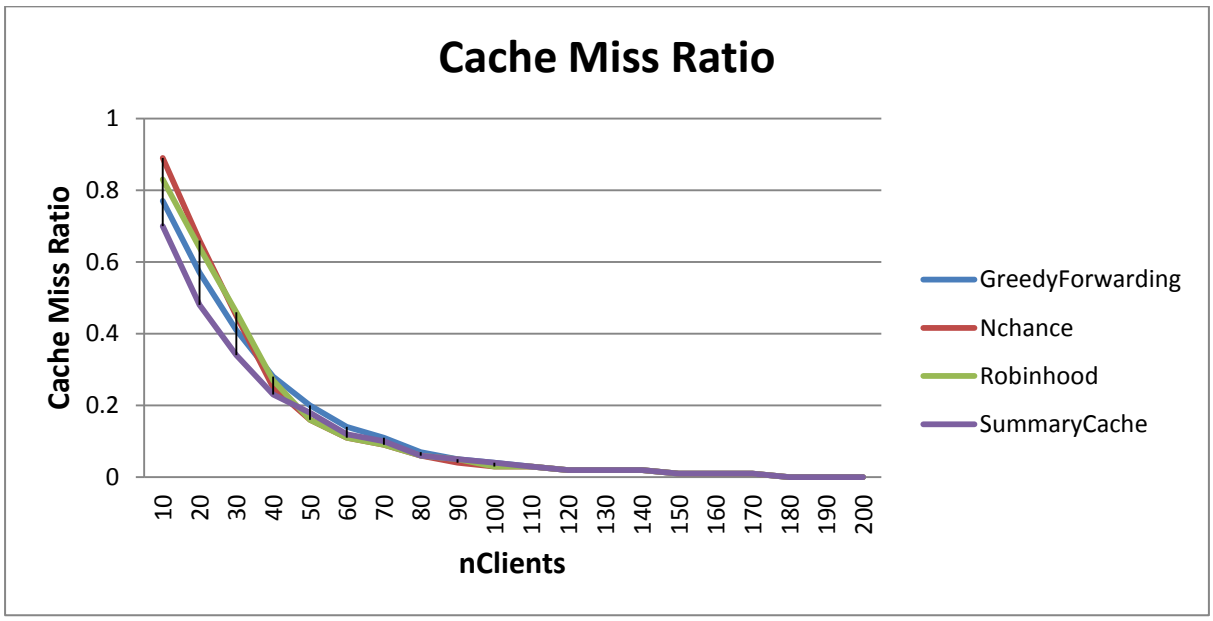
size goes beyond total disk size. Thus, ideally the average block access ticks should be constant for this scenario as every request should be served from the local/global cache. But clearly this is not the case from the graph as the average block access ticks are decreasing. This behaviour is because of the random client cache warm up. The random cache warm up makes sure that there are no duplicates in a client cache but does not guarantee that each client caches distinct blocks. Following graph compares the unique blocks cached during the experimental setup with the ideal distribution.




The difference between the points on these plots is the duplicate blocks cached in the overall system. The total number of disk references will be at least the number of duplicate block entries. Eventually all the four algorithms will handle further disk references differently. Due to this random cache warm up the average cache access ticks becomes constant as the number of clients increases from 100 to 200. To support the above results I ran another experiment with higher number of clients. Following are the graphs:



From the above graph it is evident that beyond 100 clients pretty much every request is served by the local/global cache and disk is never referred. The graph shown below shows that beyond 100 clients Cache miss ratio tends to 0 which means no request is served from the disk.




## *Analysis*

The above results show that Summary Cache with Importance Aware Bloom Filter is better than the other three thus proving my hypothesis. This performance improvement is basically because each client on local cache miss references other client's bloom filter for set membership. This process is done for all the peers until it finds a positive and forwards the query to the respective client. In Greedy Forwarding each client maintains it own cache greedily and if there is a local cache miss then request is passed to server to check for peers and ultimately to server disk. NChance and Robinhood are the variations of Greedy Forwarding where they focus more on the management of Singlet (single cached block in all the cooperative caching). Because, in summary cache each client refers its peer before going to server, the numbers of disk references are less thus reducing the overall cache block access ticks.

## *Future Plans*

The next step of my project is to document the code, prepare final project report and poster.

## *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.**