Edge Caching in Dense Networks

Rohin Garg
160583
BTech, IIT Kanpur

Summer Project: Research Track Exploration NYU - IIT Kanpur

Under the Guidance of

Dr. Fraida Fund
NYU Tandon School of Engineering

Dr. Shivendra Panwar
Professor, NYU Tandon School of Engineering

Contents

1 Introduction

The demand for rich multimedia services over mobile networks has been soaring at a tremendous pace over recent years. The current cellular architecture will not be able to handle the projected growth in mobile traffic data, especially the increase in multimedia streaming.

A Potential solution: Edge Caching

In order to cope with the mobile data traffic explosion, mobile network operators (MNOs) deploy small cell base stations (SBS) which operate in conjunction with the macrocell base stations (MBS).

1.1 Small Cell Networks

- A way to meet unprecedented traffic demands is to deploy SCNs, which provide short-range, low-power, low-cost small base stations.
- But they are not able to solve peak traffic demands using the existing reactive networking paradigm:
- Users trafic requests must be served urgently as they come or dropped causing outages Also, a large scale deployment of SCNs is not viable due to site acquisition, installation and backhaul costs.

1.2 The Role of Proactive Caching in 5G Wireless Networks

It involves:

- It involves network nodes exploiting users' context information, anticipate users' demands and use predictive abilities to save resources and cache potential future requests.
- Intelligent pre-allocation of resources and effective scheduling of user requests.
- Leveraging the powerful processing capabilities and large memory storage of devices enables network operators to proactively serve predictable peak-hour requests during off-peak times.
- Minimization of Inter-ISP traffic
- Minimization of Intra-ISP traffic
- Minimization of content access delay for users

A Model discussed by "Cache in the Air:Exploiting Content Caching and Delivery Techniques for 5G Systems":

2 Problem at Hand

To analyze the effect of network densification on edge caching performance for different for different caching methods, by:

- Conducting experiments on a simple topology, to verify data given in existing literature.
- implementing realistic content request patterns.
- Creating new network model, dense, and running experiments on it.

I wanted to see the results of existing algorithms on different social scenarios.

2.1 Metrics

I have some metrics based on which I can calculate the efficiency of the design:

- Satisfied requests.

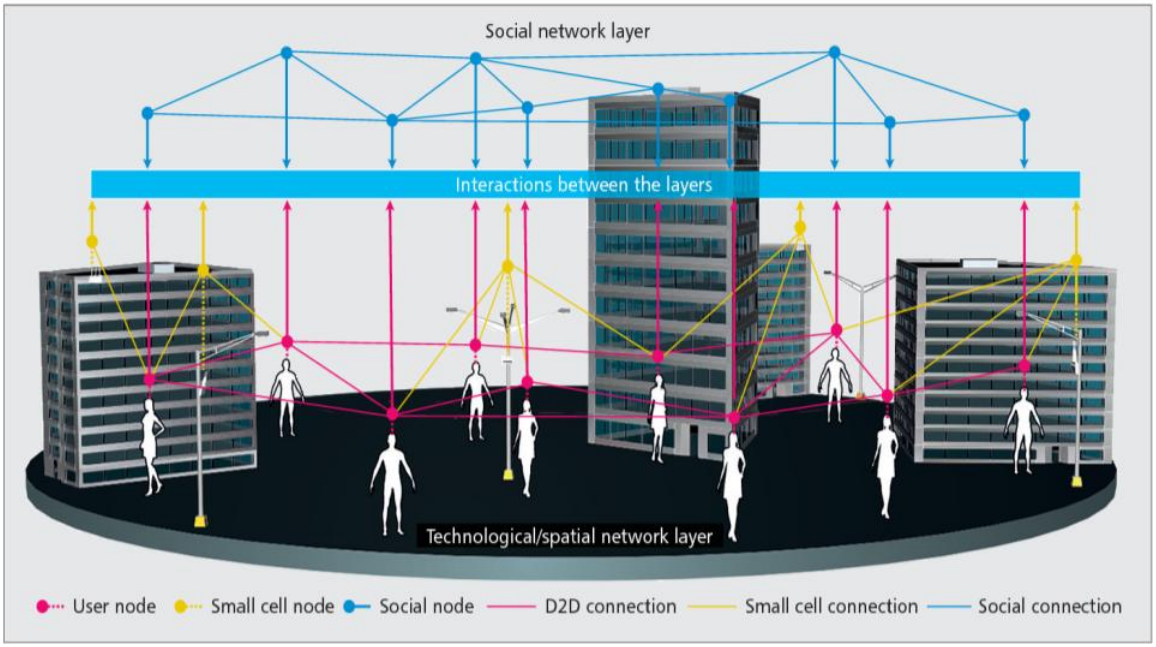


Figure 1: An illustration of an overlay of socially interconnected and spatial network [1]

- Requests routed to MBC and ISP: should be minimized.
- Hit Rate
- Delay in retrieving user requested content
- Backhaul Load

3 Experiment 1

Used a simple network topology to generate data regarding *Hit-Rate* at any cache unit (not necessarily the first one hit), by varying total cache size and SCN cache size. This is *not* a dense network.

Topology used:

I modeled a simple version of a multi-level cache system for a single 'location', inspired from [5].

Various Parameters that had to be decided

- Total Content size was fixed.
- Constant upper limit on individual object size.
- Cache size in Small Cell was set much lower than Core cache size.
- Some files were treated much more popular than most others, some not as much, and some were requested rarely.
- Transmission capacity of each small cell was kept same.

Observations

Ht rate v/s Total cache size.

The decreasing slope can be explained as increasing the cache size will only result in caching contents that are not popular/retrieved only once. They do not contribute to hit-rate.

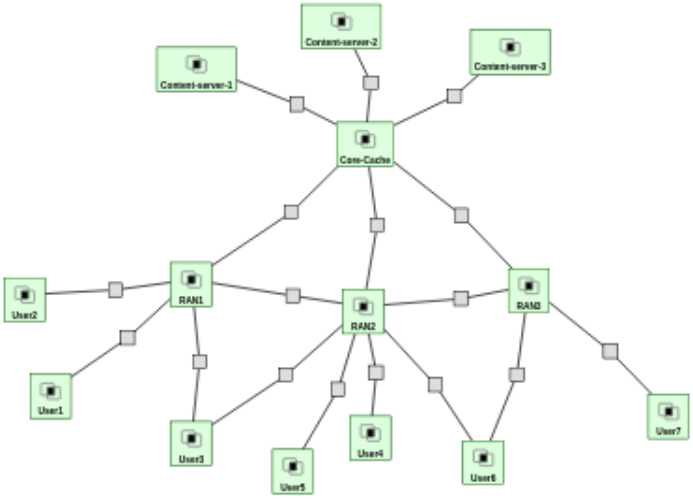
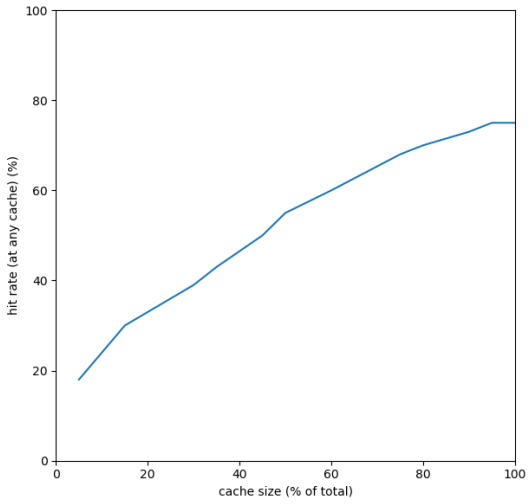


Figure 2: Topology Used



3.1 Limitations

- Not a very realistic case
- Small cells were not able to communicate properly among themselves.
- Issue of redundancy of cache: in Core and Small Cells
- 1 User only contacted 1 Small cell.
- File popularity was constant; needs to change dynamically, and so do the cached files

4 Realistic Request Patterns

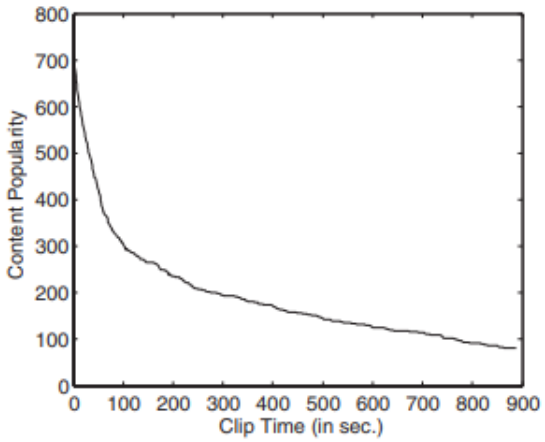
User requests are directly influenced by content popularity. A probabilistic model based on real data has been discussed in a lot of literature [2][3][4].

A popularity model based on Zipf-like distribution:

1. The access frequency (usually the parameter used to define popularity) of the r^{th} most popular file is proportional to $1/r^{\alpha}$.
2. If the frequencies of files and the corresponding popularity ranks are plotted on a log-log scale, a Zipf-like distribution can be fitted by a straight line.

Here, α is a parameter that controls the skew-ness of this distribution.

Content-Popularity, in general, decreases with time-duration, which can be implemented in the form of size.



Probabilistic Model (simple) for content popularity:

- Let N be the total number of web pages in the universe.
- Let $P(i)$ be the conditional probability that, given the arrival of a page request, the arriving request is made for page i.
- Let all pages be ranked in order of popularity, s.t page file i is i^{th} most popular page (rank).
- $P(i)$, defined for $i = 1, 2, ..., N$, has a "cut-off" Zipf-like distribution given by:

$$P(i) = \frac{\Omega}{i^{\alpha}}$$

where

$$\Omega = \left(\sum_{i=1}^{N} \frac{1}{i^{\alpha}} \right)^{-1}$$

It was found that $0.75 < \alpha < 0.82$ matched data from many popular websites.
Although, it changed for video content, with the distribution getting more skewed, and α going up to 1. Popular video sites like youtube had an inconsistent distribution, $\alpha = 1.2$ for higher ranked videos, and $\alpha = .5$ for lower ranked videos.

5 Experiment 2

For the next experiment, I used a more dense topology, with more interconnected Small Cells at a location.

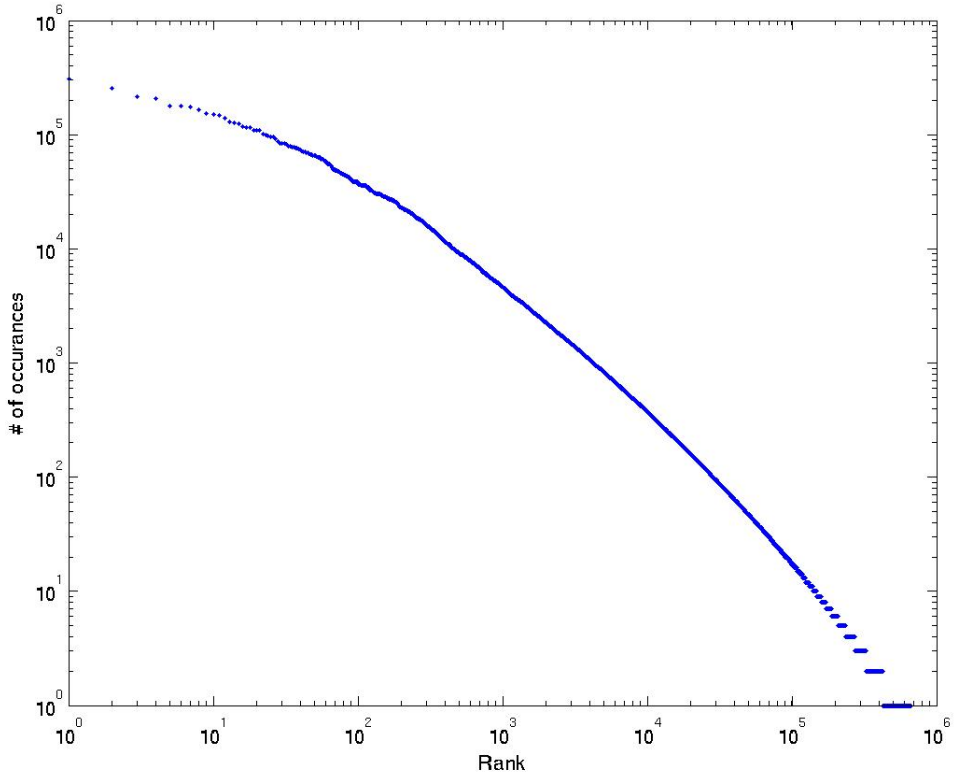


Figure 3: Zipf distribution on a log-log scale, with $\alpha = 0.80$

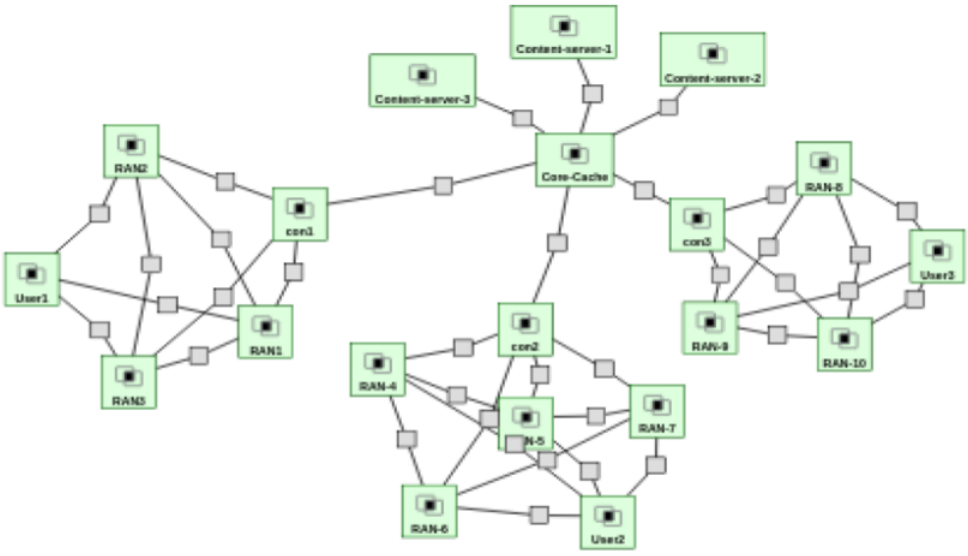


Figure 4: A dense topology

5.1 Set up and Constraints:

- Files stored in the content server:
  Majority (55%) of them were of size 150 KB to 1.5MB.
  Some (35%) were very small files $<$ 150 KB.
  Very few (10%) were large files, $<$ 5 MB.

- Max File size :
  Small cell: 1 MB
  Core cache: 3 MB

- Capacity of Cache storage:
  Small Cell: 500 MB
  Core Cache: 2000 MB.

- 2000 requests per each user.

- Every user followed the same request pattern. : Zipf - like distribution, $\alpha = 0.8$.

- Pre-cache: 1st 300 ranked files.

No. of users per Small Cell kept changing for each experiment phase, so did the no. of Small cells.

5.2 Implementation

I implemented virtual users for all the locations. The number of virtual users and small cells in each location were variables, which controlled the density of the network.

Cache Placement Policy: Leave Copy Down.

I initially ran my experiment for 2 different sets of files:
Ranked Randomly,
Ranked such that low size files are (mostly) higher ranked.

The second way makes it more realistic, so I proceeded with it for the rest of the experiment.

The overall Hit Rate (Hit rate in both Small cells and Core) was similar in both the cases.
But Local Hit Rate (Small cell hit rate) were significantly different.

Squid was used to implement and manage Cache systems on virtual servers provided by **GENI**.

Realistic Request patterns were implemented in Python 3.6.

```
import subprocess
import random
import os

n=10500   # total number of files.
a=0.80    # Parameter for Zipf distribution.


sum=0.0

for i in range(n):
  # clculationg cumulative distribution
  sum=sum+(1/float((i+1)**a))

cdf=[]
prob=[]


cmd='wget http://content-server-1/test1/'  # directory on content server.
chuser="'http://ran"


for i in range(n):
  prob.append(round((1/(i+1)**a)/sum,6))  # probability for each file
  if i !=0:
    cdf.append(cdf[i-1]+prob[i])       # cumulative distribution
  else:
    cdf.append(prob[i])

x=random.randint(1,1000000)
for j in range(n):
  if j==0:
    if x <= cdf[j]*1000000:      # checking in which file's
                  # probability range does x fall in.
      subprocess.call(cmd+str(j+1), shell=True)
  else:
    if x <= cdf[j]*1000000 and x > cdf[j-1]*1000000:
      subprocess.call(cmd+str(j+1), shell=True)
```

5.3 Results

At first: 2 locations with 3 Small Cells, and one location with 4 Small Cells.
I went from 1 user per Small Cell to 5: (less dense to more dense)

Users per SC	Overall Hit Rate(%)	Core Hit Rate(%)	Local Hit Rate(%)
1	65	38.6	43
2	63	40.1	41
3	61	42.2	39
4	59	43.5	34
5	55	44.8	31

Then I varied the number of Small Cells at each location, kept no. of users 10 at each location: went from 1 small cells to 4 at each location. (effect of small cell numbers at dense location)

Small cells	Overall Hit Rate(%)	Core Hit Rate(%)	Local Hit Rate(%)
1	65	35.5	60
2	63	38.6.1	50.2
3	59	40.6	42.8
4	56.8	42	38.3

I then tried a *direct cache system*, i.e, a direct connection between User-end-device and Core-cache, bypassing the small cells.
I varied the number of users at each location, and noted the Hit rate at Core-cache:

Users	Core Hit Rate(%)
6	67.2
7	66.9
8	67.5
9	65.4
10	64.3

6 Limitations

- Couldn't implement predictive caching. I was not able to configure the cache management system on the servers to store information and predict what files to be cached.
- Space constraints. Due to space constraints on the servers, the file sizes were kept smaller than ideal and their number was less too.
- I wasn't able to use the sibling directive in Squid: which means that the small cells at a location were not able to communicate with each other. Had this been done, hit rate theoretically would have increased .
- Only tested for Web-page content, not Video content which amounts to a large part of today's web-traffic.

7 References

[1] Living on the edge: The role of proactive caching in 5G wireless networks: Ejder Bastug ; Mehdi Bennis ; Mérouane Debbah

[2] Modeling the Content Popularity Evolution in Video-on-Demand Systems: Attila Kőrösi1, Balázs Székely and Miklós Máté.

[3] Investigation on the Content Popularity Distribution under K-Transformation in Streaming Applications: Zongkai Yang, Tai Wang, Xu Du, Wei Liu and Jiang Yu

[4] Web Caching and Zipf-like Distributions: Evidence and Implications: Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker.

[5] Cache in the air: exploiting content caching and delivery techniques for 5G systems: Xiaofei Wang ; Min Chen ; Tarik Taleb ; Adlen Ksentini ; Victor C.M. Leung