Caching in 5g networks

June 30, 2017

Ruben de Baaij
supervised by
Jasper Goseling, Berksan Serbetci
University of Twente

Abstract

Efficient ways of caching, saving files in local devices, is becoming more important. Especially with the upcoming 5g network. In this paper a way of distributing files over networks of caches is modeled and analyzed.

1. Introduction

Internet traffic becomes more and more busy every year. More files are being requested and shared constantly. The existing digital infrastructure is struggling to keep up, and with the upcoming 5g network the demand of files increases even more. This is why a lot of research is going on to find new ways of transferring files. One of the methods to deal with the huge amount of file requests is the use of caches.

Caching is temporarily storing much requested data inside a memory devices called caches. When a file is requested it will be answered by a cache in which the file is stored, it will send the file to the user that requested it. This is faster than getting the file from the original server. Saving files in caches is a way to cut out a lot of internet traffic and more file requests can be answered.

Caches, also called base stations (BS), can be located anywhere around a user. Often a user is able to connect to multiple caches in the area. By an efficient distribution of files over the caches these multiple caches in range can be taken advantage of. There is no need to store the same file in every cache a user can connect to. It is enough to answer the request when a file is stored in just one of the caches in the area.

To find such a distribution of files a lot of questions come up. Which files have to be stored in which cache? In this paper the probability that a request cannot be answered will be minimized. So the probability a user will recieve the file he requests will be optimized.

2. The Model

To find the optimal distribution of J files over N caches the following function $f(\mathbf{B})$ is used as an objective function in a mixed integer optimization system. The function gives the probability a users' file request is not answered using the file distribution matrix $\mathbf{B}$.

The vector a represents the probabilities a file is requested. These probabilities are generated using a zipf distribution (1) with parameter γ.

$$a_j = \frac{j^{-\gamma}}{\sum_{j=1}^{J} j^{-\gamma}} \tag{1}$$

This is possible because a lot of internet traffic is caused by a relative small subsection of all the files available. This zipf distribution can been seen as some sort of popularity distribution. The more popular the file, the higher the probability it will be requested.

The vector p represents the probabilities of a user being in an area where he can connect to the caches in s. Θ is the set of all the combinations of caches a user can be in range of at once.

Furthermore $b_j^{(m)}$ indicates if file j is stored in cache m. It equals 1 if the file is saved, and 0 if it is not saved. These indicators are stored in the $N - by - J$ distribution matrix $\mathbf{B}$. In table 1 of the appendix an overview of the defined variables is given.

$$f(\mathbf{B}) = \sum_{j=1}^{J} a_j \sum_{s \in \Theta} p_s \prod_{m \in s} (1 - b_j^{(m)}) \tag{2}$$

Minimizing this function will give the optimal distribution matrix $\mathbf{B}$. This optimization system is mixed integer because of the product $\prod_{m \in s} (1 - b_j^{(m)})$ which can be either zero or one.

Caches are limited in the amount of files they can store. A cache cannot store every file avaible, therefore every cache has the capacity to store K files. This is why the objective function has to be minimized subject to the following equality constraint for every cache.

$$\begin{aligned} & minf(\mathbf{B}) \\ s.t. \quad & b_1^{(m)} + \ldots + b_J^{(m)} \leq K \end{aligned} \tag{3}$$

2.1. Convexity

Solving this optimization problem is not yet possible because the model is not convex. Therefore the following variable is introduced.

$$Z_s = \prod_{m \in s} (1 - b_j^{(m)}) \tag{4}$$

This variable Z_s equals 0 if file j is stored in one or more caches in s.

Such a variable can be written differently, which will yield the same result, but in a convex optimization system.

If file j is not stored in any caches in s then all $(1 - b_j^{(m)})$ terms are 1, and so the following equation holds.

$$\sum_{m \in s} (1 - b_j^{(m)}) = |s| \tag{5}$$

From (5), if file j is not stored in any cache of s.

$$\sum_{m \in s} (1 - b_j^{(m)}) + 1 - |s| = |s| + 1 - |s| = 1 \tag{6}$$

Now if file j is stored in $k \geq 1$ caches in s then the next equations hold.

$$\sum_{m \in s} (1 - b_j^{(m)}) + 1 - |s| = |s| - k + 1 - |s| \tag{7}$$

$$|s| - k + 1 - |s| = -k + 1 \leq 0 \tag{8}$$

So from (7) and (8), if file j is stored in one or more caches in s.

$$\sum_{m \in s} (1 - b_j^{(m)}) + 1 - |s| \leq 0 \tag{9}$$

And so (4) can be written written as follows

$$Z_s = max\{0, \sum_{m \in s} (1 - b_j^{(m)}) + 1 - |s|\} \tag{10}$$

Because written like this, (10) has the same properties as (4).

$$Z_s = \begin{cases} 1 & \text{If file } j \text{ is not stored in any cache of } s \\ 0 & \text{If file } j \text{ is stored in one or more caches of } s \end{cases}$$

Because of the new Z_s the objective function of the model now satisfies more constraints and the optimizations system is now convex. It is now solvable.

$$f(\mathbf{B}) = \sum_{j=1}^{J} a_j \sum_{s \in \Theta} p_s Z_s \tag{11}$$

$$\begin{aligned} & minf(\mathbf{B}) \\ s.t. \quad & b_1^{(m)} + \ldots + b_J^{(m)} \leq K \end{aligned} \tag{12}$$

3. Solving the model

The model is solved in MATLAB, using a software package called 'cvx'. This package is able to solve all kinds of optimization systems, using different solvers for different forms of systems.

To solve the model, a mixed integer optimization system, the solver Mosek is used. The objective function and it's constraints are both given as an input.

The code written in MATLAB generates the optimal distribution matrix **B** from the users input variables. These input variables describe the amount of caches and files, and the range and capacity of every cache. From this the code generates and plots a random network of the caches, and solves the optimization system.

The locations of the caches can also be specified as input, instead of a random locations, so real networks can also be solved. This has been done for an existing network of caches in Berlin.

In figure 1 a plot of a small network consisting of six caches and eight files can be seen. Every cache has a capacity of three files. As seen in the corresponding distribution matrix (13), the files with the lowest request probability does not get stored much.

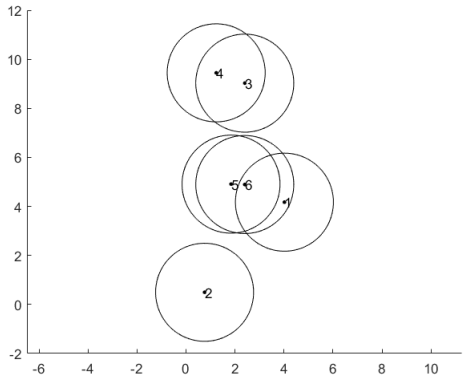

Figure 1: *A small network*

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{13}$$

In figure 2 the locations of caches of a real network in Berlin are plotted. This network consists of 62 caches, each with a capacity of three and range of 700, and 200 files. Unfortunately the network is too big to solve in an appropiate amount of time, so the results are run with less files, namely 40. The resulting distribution matrix has dimensions $62 - by - 40$, and the resulting probability a request is not answered is $0,2177$.

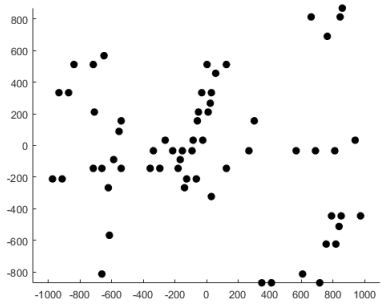

Figure 2: *Berlin network*

4. Analysis

When solving the model every entry in the distribution matrix **B** is a variable. So a network consisting of N caches and J files has $N * J$ variables. When the network is quite small the solving of the model does not take too much time, but when the network is big the runtime of the code can increase drastically. In this section there will be looked at the change of probabilities, andd also at how much the runtime increases when parameters like caches, files and range are changed.

These tests in this section have all been run with a cache capacity of three files and a range of three. (Except for figure 8.) Because of the

randomness of the cache locations, to make the result more robust, every network has been ran five times and the result is their mean.

In the appendix all the figures' corresponding tables can be viewed.

4.1. Miss probabilities

In figure 3 some networks with different amount of caches and their respective probabilities that a file request cannot be answered, so called 'miss probabilities', are graphed while adding more files to the networks.It can be seen that if a network has fewer files, the probibility of missing a file is lower. This is makes sense because there are fewer files to request.

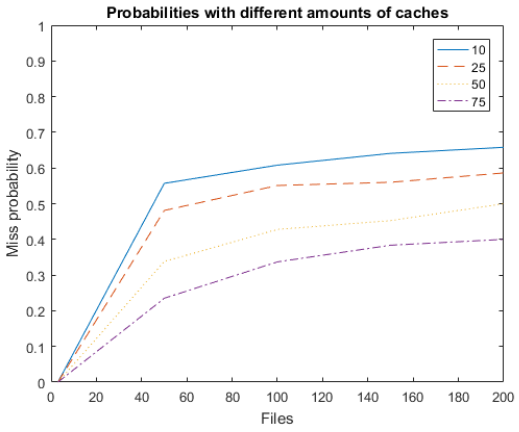

Figure 3: *Miss probabilities adding files*

When a network increases in the amount of caches the miss probability decreases. This is also seen in figure 4.

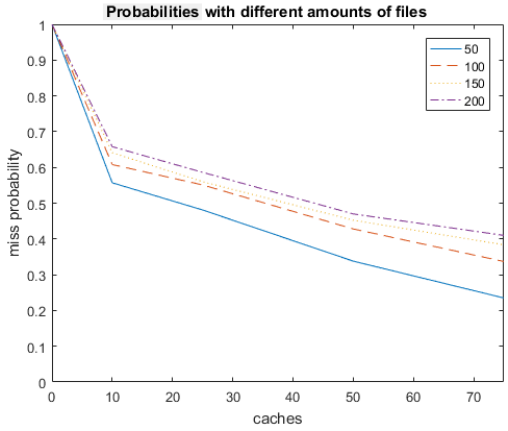

Figure 4: *Miss probabilities adding caches*

4.2. Simple distribution

Most often existing caches save only the few most popular files. Such a simple distribution matrix consists of only 1's the first K (cache capacity) columns and zeros in all the remaining columns. This way of distributing files is very inefficient when a lot of caches overlap. Many files will be saved in multiple caches in range of a user.

In figure 6 the change of the miss probability for a network with 10, 25, 50 and 75 caches and a cache capacity of three, are graphed while increasing the amount of files. The result is the same for each network because when only the first three most popular files are stored, no matter how many caches their are, in every area the miss probability is the same.

Adding cases to a network is only useful when they save more different files.

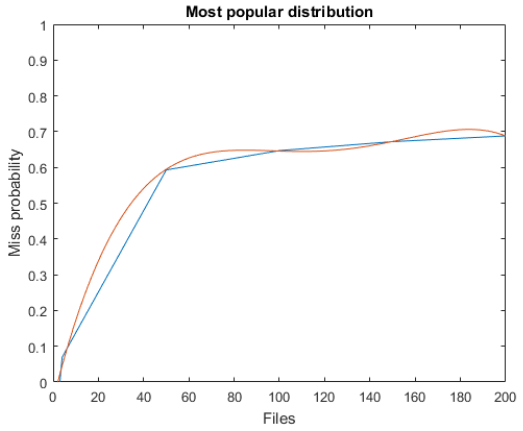

Figure 5: *Miss probabilities adding files using a simple distribution strategy*

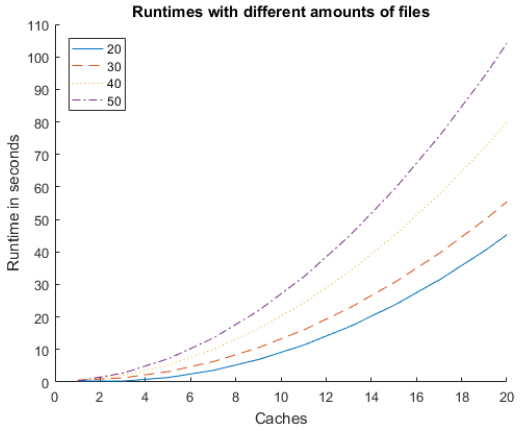

Figure 7: *Runtime increase while adding caches*

The increase of runtime when adding files to a networks seems to grow almost linear, while when adding caches to a network the runtime seems to increase exponentially.

4.3. Runtimes

For big networks the runtime of the code will be very long. In figures 6 and 7 the increase of runtime can be seen when adding files and caches, for networks with different amount of caches.

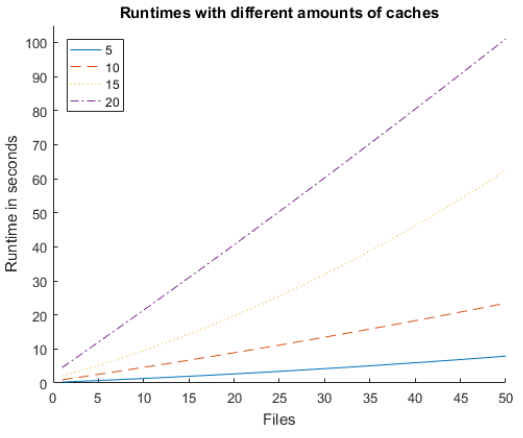

Figure 6: *Runtime increase while adding files*

The more caches in a network the steeper the increase in runtime when adding files.

4.4. Cache ranges

The range of caches is an important factor when distributing files over a network. When the cache ranges are large there will be many overlapping areas, causing more different files to be saved. For the graph of figure 8 networks with 10, 25 and 50 caches, all with 25 files, are tested.

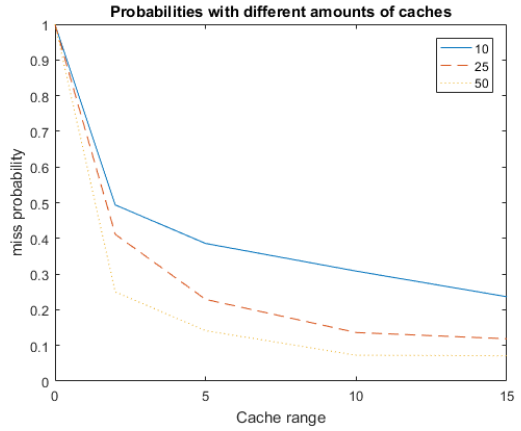

Figure 8: *Change of miss probabilities when cache range increases*

As expected the miss probabilities decrease while the cache range becomes larger.

4.5. Variables and Constraints

When running the code 'cvx' calculates the amount of variables and constraints in the optimization system. These depend on the amount of caches and files in the network, but also on the amount of overlapping areas.

For the figures in this section networks were used in which there where no overlapping areas. Every cache has their own area and no caches caused overlapping areas.

This has been done because when there are overlapping areas the variables and constraints of the system change a lot. Using random networks, causing random amounts of overlapping areas, will give many different results because of this.

In the figures 9 and 10 the increase of variables and corresponding amount of constraints, while adding more files to the networks, are graphed. Both seem to increase linear although the amount of variables increase way faster then the constraints.

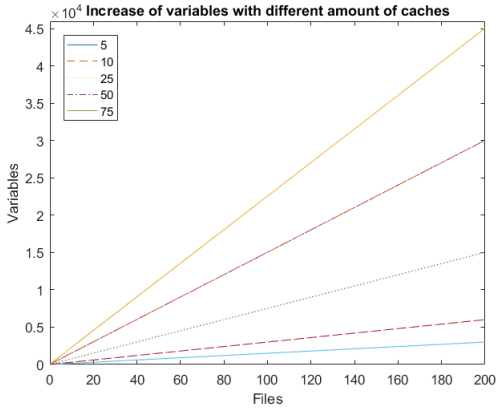

Figure 9: *Variables*

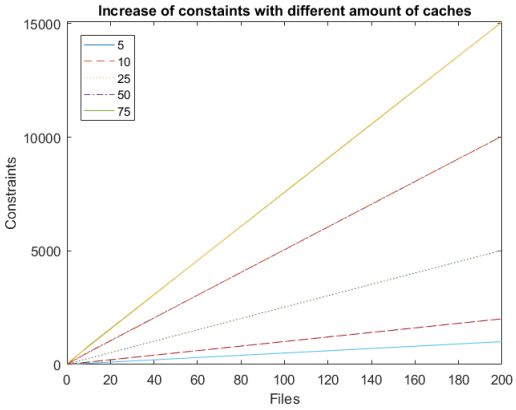

Figure 10: *Constraints*

The runtimes for generating a distribution for the non overlapping networks are given in figure 11 and 12.

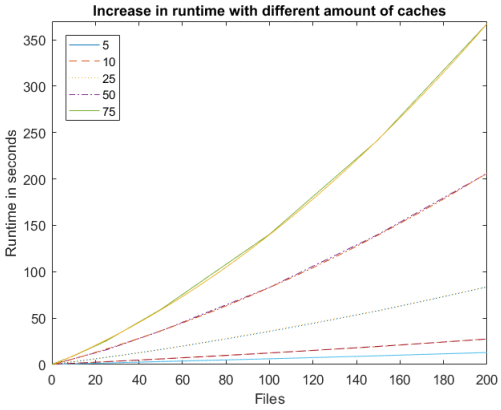

Figure 11: *Runtimes adding files*

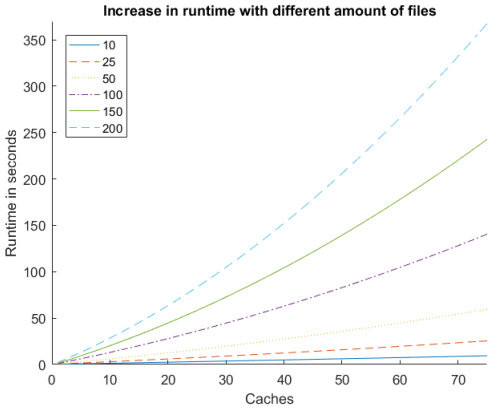

Figure 12: *Runtimes adding caches*

5. Discussion and Conclusion

By distributing files over a network of caches using the method described in this paper some internet traffic will be relieved from the current infrastructure.

Using caches efficiently does have a significant result on the amount of file requests that can be answered. The more caches, and the larger the their range, the more files can be saved and the more file requests will be answered.

Calculating a distribution for a big network gets complex really fast, but by fine-tuning the code used this will get a lot better. The variables and constraints for the model described grow fast, especially in big networks. If files will be distributed in this way cache networks in the future will be way more efficient then they are now.

References

[1] Jasper Goseling
Berksan Serbetci
Stochastic Operations Research
University of Twente
Konstantin Avrachenkov, INRIA Sophia Antipolis
A Low-Complexity Approach to Distributed Cooperative Caching with Geographic Constraints On Optimal Geographical Caching in Heterogeneous Cellular Networks

[2] Michael Grant and Stephen Boyd
CVX: Matlab Software for Disciplined Convex Programming, version 2.1
`http://cvxr.com/cvx`
March 2014

[3] Negin Golrezaei, University of Southern California
Andreas F. Molisch, University of Southern California
Alexandros G. Dimakis, Viterbi School of Engineering, University of Southern California
Giuseppe Caire, Viterbi School of Engineering, University of Southern California
Femtocaching and Device-to-DeviceCollaboration: A New Architecture for Wireless Video Distribution 2013

[4] Nicaise Choungmo Fofack, Sara Alouf
Modeling modern DNS caches 2013

[5] Arpan Chattopadhyay, BartÅĆomiej BÅĆaszczyszyn
Gibbsian On-Line Distributed Content Caching Strategy for Cellular Networks 2016

6. APPENDIX

Figure 13: *Miss probabilities change*

Caches/files	50	100	150	200
10	0.557	0.608	0.641	0.658
25	0.481	0.551	0.560	0.5862
50	0.338	0.428	0.4525	-
75	0.235	0.337	0.3836	-

Figure 14: *Comparison to most popular distribution*

Caches/files	50	100	150	200
10	0.593	0.647	0.672	0.688
25	0.593	0.647	0.672	0.688
50	0.593	0.647	0.672	0.688
75	0.593	0.647	0.672	0.688

Figure 15: *Runtime change*

Caches/files	20	30	40	50
5	2.406	4.431	6.037	7.795
10	9.532	12.576	18.658	23.413
15	21.802	29.945	45.954	62.950
20	45.947	55.641	79.329	102.515

Figure 16: *Change of miss probabilities when cache range increases*

Range/caches	10	25	50
2	0.4938	0.4114	0.2501
5	0.3858	0.2290	0.1414
10	0.3083	0.1362	0.0726
15	0.2366	0.1187	0.0711

Table 1: *Legenda*

Symbol	Description
C	Set of all files
J	Amount of files
c_j	File j
a_j	Probability file j is requested
N	Amount of caches
K	Capacity of a cache
m	Cache m
Θ	Set of all possible combinations of caches that can be in range of an area
s	Set of caches in range of an area
A_s	The area in which all caches in s are in range
p_s	Probability a user is in an area in range of the caches in s
A_{cov}	The total area all caches cover
$b_j^{(m)}$	File storage index of file c_j in cache m
$\mathbf{B}$	Storage policy matrix (NXJ)
$f(\mathbf{B})$	Probability a requested file is not found

Table 2: *Variables*

Caches/files	10	25	50	100	150	200
5	155 v 55 c	380 v 130 c	755 v 255 c	1505 v 505 c	2255 v 755 c	3005 v 1005 c
10	310 v 110 c	760 v 260 c	1510 v 510 c	3010 v 1010 c	4510 v 1510 c	6010 v 2010 c
25	775 v 275 c	1900 v 650 c	3775 v 1275 c	7525 v 2525 c	11275 v 3775 c	15025 v 5025 c
50	1550 v 550 c	3800 v 1300 c	7550 v 2550 c	15050 v 5050 c	22550 v 7550 c	30050 v 10050 c
75	2325 v 825 c	5700 v 1950 c	11325 v 3825 c	22575 v 7575 c	33825 v 11325 c	45075 v 15075

Table 3: *Runtimes*

Caches/files	10	25	50	100	150	200
5	0.7423	1.6297	3.1677	6.1373	9.3968	12.9194
10	1.3107	3.0522	6.0692	12.4735	19.3260	27.5507
25	3.0581	7.6029	15.7807	36.2222	57.8310	83.7663
50	6.1947	15.8061	36.0718	83.0759	140.2921	206.0263
75	9.3152	25.4866	59.2876	140.3925	242.3616	367.4525