Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών,
Δικτύων και Τηλεπικοινωνιών

Διαχείριση Πόρων σε Δίκτυα Περιεχομένου

Κωνσταντίνος Πουλαράκης

Επιβλέποντες:
Λέανδρος Τασιούλας
Γεώργιος Πάσχος

Αφιερωμένο στην οικογένεια μου,
Νίκο, Χριστίνα και Στέργιο

Περίληψη

Η διαρκώς αυξανόμενη χρήση των Video On Demand εφαρμογών και η διάδοση των IPTV υπηρεσιών αυξάνουν τις ανάγκες για εύρος ζώνης, ενδυναμώνουν τη δημιουργία συμφόρησης στις ζεύξεις του Internet και την εμφάνιση υψηλών καθυστερήσεων στην πλευρά των χρηστών του. Οι στρατηγικές διαχείρισης της κρυφής μνήμης (caching) παρέχουν έναν αποτελεσματικό τρόπο για να περιοριστούν οι απαιτήσεις για εύρος ζώνης, μέσα από την επιλεκτική τοποθέτηση περιεχομένου κοντά στους χρήστες αντί να τοποθετείται σε ένα μόνο κεντρικό εξυπηρετητή. Στην εργασία αυτή παρουσιάζουμε αλγόριθμους τοποθέτησης περιεχομένου με στόχο την ελαχιστοποίηση του της έντασης της κυκλοφορίας δεδομένων που εξυπηρετούνται από τον κεντρικό εξυπηρετητή (αντί από τις κρυφές μνήμες), δεδομένης της τοπολογίας από κρυφές μνήμες και του προφίλ των αιτήσεων για περιεχόμενο από τους χρήστες. Χρησιμοποιείται επίσης ως μέτρο βελτιστοποίησης και το μέσο κόστος λόγω της κίνησης των δεδομένων στις ζεύξεις του δικτύου. Εστιάζουμε στο συγκεκριμένο τύπο τοπολογίας δικτύου που είναι ιεραρχικός, θεωρώντας τον κεντρικό εξυπηρετητή ως ρίζα του, και δεχόμαστε ότι επιτρέπεται η συνεργασία μεταξύ κρυφών μνημών διαφορετικών επιπέδων, αρκεί η κρυφή μνήμη που ανταποκρίνεται σε μια αίτηση να βρίσκεται στο μονοπάτι που ενώνει την αιτούσα κρυφή μνήμη και τον κεντρικό εξυπηρετητή. Αντίθετα, δεν επιτρέπουμε συνεργασία μεταξύ κρυφών μνημών του ίδιου επιπέδου ιεραρχίας. Η συνεισφορά μας έγκειται στην παρουσίαση βέλτιστων πολυωνυμικής πολυπλοκότητας αλγορίθμων τοποθέτησης αντικειμένων στο παραπάνω μοντέλο, θεωρώντας ότι όλα τα αντικείμενα είναι ίδιου μεγέθους, κανονικοποιημένο στη μονάδα. Επίσης, εξετάζεται το πρόβλημα της από κοινού επίλυσης του προβλήματος τοποθέτησης περιεχομένου και ανάθεσης των κινητών πελατών στους κόμβους-φύλλα της ιεραρχίας κρυφών μνημών. Παρουσιάζουμε ένα βέλτιστο αλγόριθμο,

για το πάνω πρόβλημα, υποθέτοντας ότι δεν επιτρέπεται συνεργασία μεταξύ των κρυφών μνημών. Για την περίπτωση που επιτρέπεται συνεργασία μεταξύ των κρυφών μνημών, προτείνουμε ένα νέο ευριστικό αλγόριθμο που λύνει πρώτα το πρόβλημα της ανάθεσης των πελατών στις κρυφές μνήμες-φύλλα της ιεραρχίας χρησιμοποιώντας την τεχνική της συσταδοποίησης και στη συνέχεια δεδομένης αυτής της ανάθεσης βρίσκει τη βέλτιστη τοποθέτηση περιεχομένου με βάση τον βέλτιστο αλγόριθμο που προτείνουμε. Αποτελέσματα από τις προσομοιώσεις των αλγορίθμων που εισάγαμε παρουσιάζονται, ώστε να φανεί η αποτελεσματικότητα των νέων αλγορίθμων που εισάγαμε απέναντι στους προκατόχους τους, όσον αφορά την απόσταση της λύσης που βρίσκουν από τη βέλτιστη, την ποιότητα της λύσης σε περιπτώσεις αλλοιωμένης πληροφορίας εισόδου, καθώς και τη πολυπλοκότητα και το ρυθμό σύγκλισης τους.
Το κύριο σώμα της διπλωματικής εργασίας ακολουθεί (ως παράρτημα) στην αγγλική γλώσσα.

Content placement and client assignment algorithms in hierarchical cache topologies when inter-level cooperation is allowed

Abstract

Popularity of VoD applications and spread of IPTV services increase the bandwidth demands, enforcing the creation of congestion in Internet's links and the experience of big delays to the users. Caching strategies provide an effective way for eliminating the bandwidth requirements by collectively placing content close to users instead of storing it in a central server. In this work, we present cache management algorithms aimed at minimizing the traffic volume that is not served by caches, given the cache topology and the profile of users' requests for objects. We focus on a hierarchy of distributed caches, where inter level cache cooperation is allowed and present an optimal content placement algorithm (assuming that sizes of objects are normalized to 1) subject to the constraint that each user's requests can be replied only by its own distributed cache or by a parent node on the path to the origin server, not allowing any cooperation between the distributed caches of the same level. Numerical experiments for typical popularity distributions show the performance distance between the optimal algorithm and some low complexity heuristic algorithms that are commonly applied nowadays. Finally, we present a formulation of the problem of joint content placement and assignment of clients in the leaf caches of an hierarchy of caches. We show that this problem can be solved efficiently for the case that no cooperation is allowed, and we propose a heuristic algorithm to handle the case that inter-level cache cooperation is allowed.

Contents

I. Introduction

We have witnessed a tremendous growth in the use of the World Wide Web in the past few years. Furthermore, the expansion of VoD (Video-on-demand) libraries, enforced by the popularity of user generated clips leads in huge bandwidth demands. These demands are envisioned to be further increased by the deployment of IPTV services such as live television and time-shifted programming. Instead of the traditional TV networks that use broadcast sessions, in this kind of applications there must be a unicast session for each video transmission to each single user, increasing the total bandwidth demands.

In the previous conditions it is clear that efficient content delivery to users is a challenging issue. Most effort has been dedicated to the study and development of content network as an answer to this challenge. We identify five important applications in this context:

- Server farms (mirroring)
- Web caching
- Content Distribution Networks
- Peer to peer
- Cooperative web caching

A shortly description of each of the previous applications follows:

Server farms (Mirroring)

In this type of content networks the content of an origin server is all copied to other mirror servers near the origin server. This allocation is static over time, thus creating the impression that the group of servers is actually a single origin site. The requests to the origin server are load-balanced across all servers in the group and automatic routing of requests away from servers that fail is performed. However, server farms do nothing about problems due to network congestion, or to improve latency issues due to the network overuse. Figure 1 visualizes the above description.

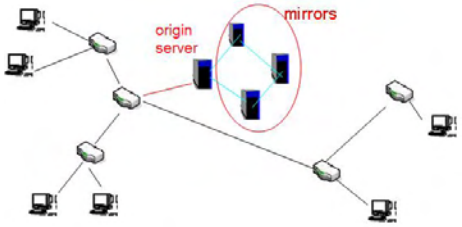

Figure 1

Web Caching

At this approach objects of more than one origin servers are copied in caches near to the users. Caches store most frequently or most recently requested content in a passive uncontrollable way. If the requested object is not contained in the cache then the request is redirected to the origin server. Web caching is typically be employed by an Internet Service Provider (ISP) for the benefit of users accessing the Internet. The benefit for the

ISP is that a fraction of content requests is absorbed by the caches, decreasing the level of congestion in the network and the high processing power needed of the central server and the benefit for the internet users is the experience of lower delays. Figure 2 illustrates this capital.

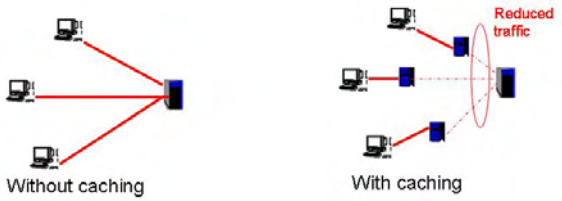

Figure 2

Content Delivery Networks (CDNs)

A CDN is a collection of content servers scattered across the network that try to absorb a fraction of work load of origin servers by cooperatively delivering content on behalf of them. Clients get the content they requested by the nearest content server found by the DNS servers. Achieves better performance and reliability than caching. Responsible of the operation of a CDN is its Content Administrator. Clients of CDN are not the ISPs as they were in caching but content providers. Unlike web caching, where content replication is decided according to the requests that arrive in caches at the last time period, in CDNs content replication is done in a deterministic way decided by a content manager. Next figure shows an example of CDN.

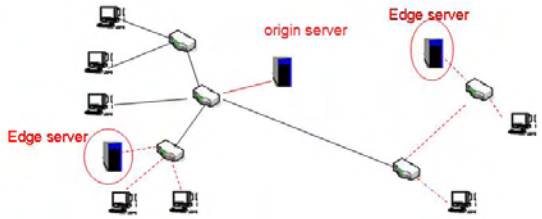

Figure 3

P2P systems

A P2P system is composed by PCs at the edge of the network which are both suppliers and consumers of files. A Peer exploit resources (bandwidth rate, disk space, CPU power) of a large collection of other peers in order to deliver a content. At this type of system scaling is feasible, enhancing the benefits of this choice. In order for a peer to find out the peers that hold the requested object directory services are needed. Examples of P2P systems are: Napster, Gnutella, Freenet, FastTrack.

Cooperative Caching

Caches at different sites in the network can cooperate in order to improve total performance. In contrast with traditional caching, a cache can serve anothers cache requests.

Caches can cooperate in order to perform storage decisions. It is needed a slightly more controlled environment where the content placement can be actively managed. In the following example clients 1,2 request in average both the blue and red file. However each of two nearby caches owns storage space to hold only one of the two files. Clearly, there is incentive to cooperate in order to alleviate the requests to the remote origin server.

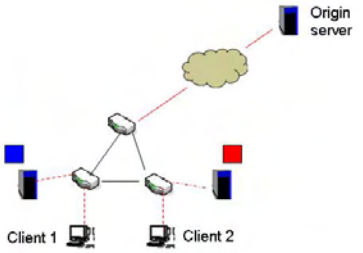

Figure 4

Given the dimensions of the caches and the sites in the network that they are placed, the design of efficient caching strategies comes mainly to the content placement problem. The content placement problem asks for selecting the contents that have to be placed at each of the available caches in order to optimize the performance metric. However, another problem that needs to be addressed is the accurate prediction of the users' demand rate for each different piece of content. This demand rate can be considered to be constant or (most realistically) to change across time. In the second case a strategy for replacement of objects in caches can be periodically applied as a best response to the change of demand rates. Besides that efficient routing of object requests can further increase the system performance. A commonly applied routing strategy is the one selecting the nearest cache that holds a replica of the object to respond to the request. Finally, for supporting additionally "write requests", i.e. requests for updating an object, a consistency mechanism is needed in order to spread the content changes to all the caches that contain a replica of this object.

In this work, we studied cooperative caching and considered the following mode of operation: Every content request is first submitted to the subset of caches that can be accessed by the user that generated this request. If content is found to be placed in any of these caches we call that a hit is performed, otherwise this request is handled by the data center and we call that a miss is performed. We focused on the content placement problem in an hierarchy of caches, when inter-level cooperation is allowed, willing to minimize the probability of miss, i.e. a user requests an object which is not found in any of the accessible caches. This problem is equivalent to the one of maximizing the fraction of requests for objects that is absorbed by the set of caches. We assume that demand estimates are given and remain constant in time, cache topology is known and the dimensions and locations of the caches in the network are constant and known.

The rest of the paper is organized as follows: We review related work in Section II and introduce our system model in Section III. Then, in Section IV we introduce a polynomial algorithm that solves the problem of optimal content placement in terms of minimizing

probability of accessing origin server for a two-level hierarchy of caches, where there are two leaf caches and a parent cache and inter-level cooperation is allowed. In Section V, we propose an optimal algorithm for the previous problem for any number of leaf caches and describe low complexity sub optimal algorithms. We extend the previous algorithm in order to solve the problem of minimizing average access cost of content requests in Section VI and for any number of levels of hierarchy in section VII. Section VIII deals with the problem of joint content placement and client assignment in hierarchical cache topologies. Simulation results of the performance of proposed algorithms are provided in section IX. Section X concludes the paper and offers directions for future research.

II. Related work

Previous work on cooperative caching can be categorized according to the assumptions about system model done. A great effort is done assuming static system model as well as assuming dynamic characteristics of the topology and/or user request patterns. Additionally, some researchers focused on the full replication case, i.e. the case that there is only one unit of content of an origin server that must be efficiently placed in the network whereas other treated the partial replication case where there are many pieces of content that must be spread across the network. A great attention is given to the study of hierarchical topologies assuming cooperation is allowed between caches of the same level (intra-level) or of different levels (inter-level) of the hierarchy. Figure 5 briefs the above grouping.

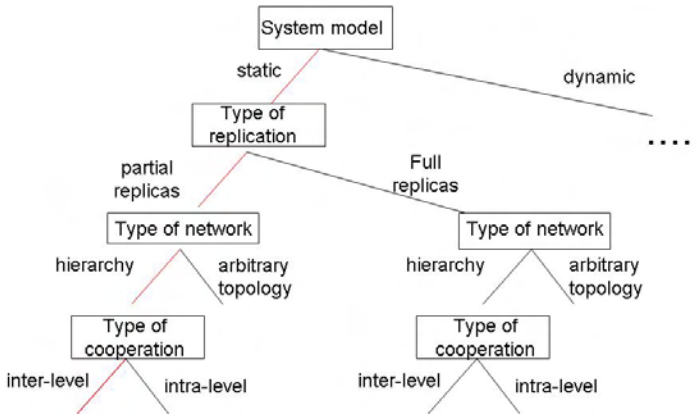

Figure 5

Full replication case has mainly been studied as a minimum k median problem of choosing M sites of the network to replicate the content of total N sites $(N > M)$. Each client is assigned to the nearest cache, that has chosen to store the replica, incurring a cost that is proportional to it's request rate and the distance (number of hops, delay due to the capacity of links..) between them. However the problem of selecting M centers so as to minimize the sum of the assignment costs is NP-hard and approximation algorithms are often used.

B. Li, et al.[1] studied the full replication case and presented an optimal algorithm for the content placement problem on tree topologies. They assumed that requests generated at a tree node follow the direct path to the root node (origin server) until they find the cached content. Thus routing of requests is predetermined and a request can not reach caches of sibling nodes. The problem of optimally choosing the M sites of the N sites of the tree to place the caches that hold the replica was formulated as a dynamic programming problem of complexity $O(N^3M^2)$.

L. Qiu, et al. [2] proposed three heuristic algorithms for the above problem. Particularly, they proposed a greedy algorithm in which the M sites are picked one by one and at iteration $i < M$ it is placed a replica at site S_i if it yields the lowest cost in

conjunction with the sites already picked. In general, in computing the cost, we assume that clients direct their accesses to the nearest replica. The complexity of greedy algorithm is $O(N^2M)$. Moreover Hot Spot Algorithm is presented which sorts the N potential sites according to the amount of traffic generated within their vicinity, and places the replicas at the top M sites. $A's$ vicinity is the circle centered at node A with some radius. The complexity of hot spot algorithm is $O(N^2)$. Finally, Random Algorithm is considered according to which, M sites from N places are chosen randomly. The complexity of random algorithm is $O(NM)$. The contribution of this paper is the comparative results os extensive simulation performed for the above algorithm including the tree-based algorithm proposed in [1]. Figure 6 shows the CDF of the relative performance of the four mentioned algorithms for a large number of experiments in random not-tree topologies. By relative performance they denoted the ratio of the performance of a heuristic algorithm to a not achievable lower bound of the optimal cost determined by a super optimal algorithm that solved the linear relaxation of the problem by using a subgradient method as described in [3]. Greedy algorithm is better than hot spot algorithm which is better than tree-based algorithm which is better than random algorithm. The non optimality of tree based algorithm is because the topologies picked were not trees.

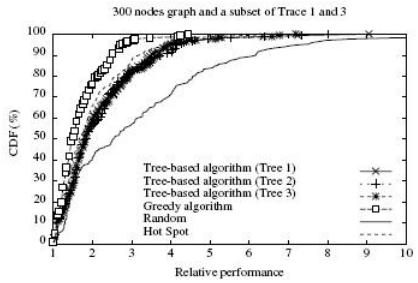

Figure 6

Partial replication case has attracted a great interest of research community over the last decade as the increasing popularity of huge video files spread across the internet revealed that storage capacity of caches is a bottleneck to the system performance and should be efficiently used. Thus, replicating only the parts of files that are usual requested instead of the whole file can increase performance. In order to describe the mathematical formulation of the PRP we use the notation organized in table 1.

Symbol	Description
O	set of objects
C	set of caches
O_j	object j
C_i	cache i
$\|O_j\|$	the size of object j
$\|C_i\|$	the storage capacity of cache i
x_{ij}	indicates if object j is stored in cache i
λ_i	the average rate of external requests generated at cache i
p_{ij}	the probability that a request generated at cache i is for object j
d_{ik}	the distance/cost between cache i and cache k
d_{i0_j}	the distance/cost between cache i and origin server of object j
K_j	the subset of caches that have stored object j

Table1.

The partial replication problem (PRP) can be stated as follows: "For a network of N caches each with different storage capacity, replicate M objects at the caches such that it satisfies the storage constraints and also minimizes the average access cost. (we assume that each request is satisfied by the nearest cache that has stored the requested object and there is an origin cache C_{0_j} that has initially stored object O_j)".

$\min_x \sum_{object\ j} \sum_{cache\ i} \lambda_i p_{ij}(1 - x_{ij}) \min_{k \in K_j} d_{ik}|O_j|$
subject to:

$$\sum_{object\ j} x_{ij}|O_j| \leq |C_i|, \forall cachei$$

$$x_{ij} \in \{0, 1\}$$

Specifying the exact form of the optimal content placement policy for the PRP appears to be intractable problem in general, as it is an NP-complete problem. That is because PRP is NP, as given a file placement x and a target cost D, we can verify in polynomial time whether the placement results in an average cost of less than D. Besides that PRP is NP-hard as it is proven below:
Consider the special case where $|C_1| = C$, $|C_i| = 0\ \forall i \neq 1$, $\lambda_i = \lambda\ \forall i$, $p_{ij} = p$, $\forall i$, $\forall j$. We define the utility of placing an object j in cache 1 as $U(j) = \sum_{cache\ i}(d_{i0_j} - d_{i1})$
Given a target decrease in cost D' we now ask if there is a set of objects J' such that: $\sum_{j \in J'} |O_j| \leq C$ and $\sum_{j \in J'} U_j \geq D'$.
By the above discussion it is seen that the problem is identical to the knapsack problem which is a known np-complete problem.

Various heuristic algorithms for solving PRP has been presented in the last years. In [4] authors presented the following four heuristic content placement algorithms:

Random
While there is free space in any cache pick uniformly a cache i and an object j. If i has

not already stored j and has enough free space to store it, put j in i. Else, pick a new pair i,j.

Popularity

Each cache stores the most popular objects according to its clients, as the storage constraints allow.

Greedy-single

Each cache i stores the most weighted objects, as the storage constraints allow. The weight of object j for cache i is equal to $W_{ij} = p_{ij}d_{i0_j}$ The disadvantage of this method against the two later mentioned is that it requires that each cache has information about the network topology (which is the origin server for each object and what are the characteristics of each link to the path of the lowest cost between the cache i and the origin server O_j). In case that there is only one origin server that holds all the objects then the resulting content placement of Greedy-single is the same to the one achieved by popularity algorithm.

Greedy-global

It inserts objects in caches one by one according to the following: While there is free space in any cache an administrator calculates a weight for each pair of cache i and object j. This weight is $W_{ij} = \lambda_i p_{ij} min_{k \in K_j} d_{ik}$, where K_j is the set of caches (including origin server O_j) that have stored object j until that step of the algorithm. Then administrator puts object j^* in cache i^* where $(i^*, j^*) = argmax W_{ij}$.

Figure 7 shows the relative performance of the above heuristics for increasing values of cache sizes as simulated in [4]. We observe that Greedy-global is better than greedy-single which is better than popularity which is better than random algorithm. However, the computation requirements needed for these algorithms is inverse proportional to the above ranking. Besides that, we observe that the performance of all the above heuristics except random is better when the zipfian parameter of the request rates is larger, as a large zipfian parameter means that a small number of objects generate a large amount of requests.

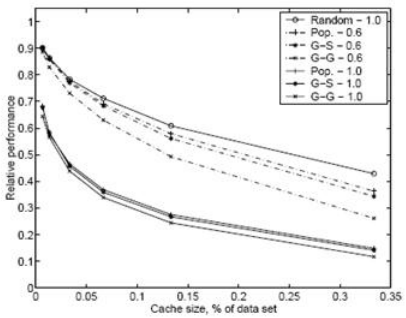

Figure 7

Another Heuristic for PRP problem was proposed in [5] and a briefly presentation of it is the following:

First calculate the total number of replicas of each object j in all the caches proportional to its total popularity from all clients. Particularly for object j set $\omega_j = \lceil \frac{\sum_{cache\ i} |C_i|}{O_j} P_j \rceil$, where P_j is the sum of the probabilities of all the clients to request object j. Then sort objects according to their aggregate popularity P_j and for each object O_j find the ω_j caches that will store it by solving a capacitated constraint p median problem. The solution to each of the previous p median problems is attained by an appropriate approximation clustering algorithm.

Simulation results in [5] showed that this algorithm is better than Random as figure 8 shows (the green line is the relative performance achieved by random algorithm and the other two lines correspond to the performance of the discussed heuristic for two values of zipfian parameter for probability of request vectors for increasing values of cache sizes):

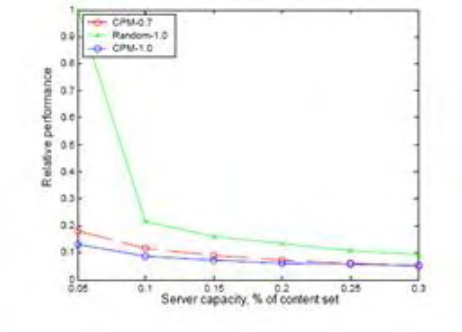

Figure 8

Ending the discussion to the partial replication case we shortly describe two other approaches: the A-star algorithms[6] and the genetic algorithms[7]. In an A-star algorithm it is constructed a tree, of which the root represents the empty cache assignment, intermediate tree nodes represent partial solutions and leaf nodes represent the complete solutions of the PRP problem. Each node n of the tree has a cost f(n)=g(n)+h(n), where g(n) represents the search path cost from root until n and h(n) is a lower bound estimate of the cost from n to a leaf node. Starting from the root of the tree, algorithm moves to the node n with the lowest cost f(n) and places in cache i object j, specified by the node n. Pruning techniques can be used to speed up the algorithm in trade off algorithm's optimality. In a Genetic Algorithm a set of binary vectors represent the assignment of objects to caches called chromosomes. Starting with a finite population of chromosomes a genetic algorithm iteratively creates new chromosomes by the operations of crossover, mutation and selection. The best chromosome (that corresponds to lowest cost assignment) replaces the worst chromosome. This operation continues until the algorithm converges.

Great effort has been done in studying and optimizing control of caches in hierarchical topologies. Particularly, for the PRP problem a significant number of studies analyzed mathematically PRP in the case that caches can cooperate by storing and serving objects that clients at other caches request. We distinguish two types of cooperation, one that

the only cooperation that is allowed between caches of the same level in the hierarchy (intra-level) and the one that cooperation is allowed between caches of different levels in the hierarchy (inter-level). Below it is shown a description of the formulation of PRP problem in two-level hierarchies in terms of each type of cooperation.

Intra-level cache cooperation PRP in a two level hierarchy is shown in figure 9.

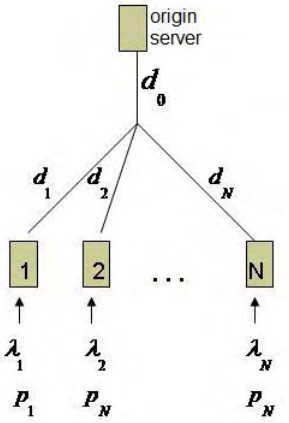

Figure 9

This problem can be equivalently seen as the problem of maximizing average cost saving instead of minimizing average access cost of client requests. The problem formulation is the following:

$\max_x \sum_{object\ j} \sum_{cache\ i} (\lambda_i p_{ij} |O_j| x_{ij} (d_0 + d_i) + \sum_{cache\ i' \neq i} \lambda_i p_{ij} |O_j| x_{ii'j} (d_0 + d_i - d_{ii'}))$

subject to:

$$\sum_{object\ j} x_{ij}|O_j| \leq |C_i|, \forall \text{ cache i}$$

$$x_{ii'j} \leq x_{i'j}, \forall \text{ cache i, } \forall \text{ cache i' } \neq i, \forall \text{ object j}$$

$$x_{ij} + \sum_{i' \neq i} x_{ii'j} \leq 1, \forall \text{ cache i, } \forall \text{ cache i' } \neq i, \forall \text{ object j}$$

$$x_{ij} \in \{0,1\}, \forall \text{ cache i, } \forall \text{ object j}$$

$$x_{ii'j} \in \{0,1\}, \forall \text{ cache i, } \forall \text{ cache i' } \neq i, \forall \text{ object j}$$

where,

$$x_{ii'j} = \begin{cases} 1 & \text{if cache i takes object j from cache i'} \\ 0 & \text{else} \end{cases}$$

$d_{ii'}$ is not necessarily equal to $d_i + d_{i'}$

Related work on the content placement for hierarchical cooperative caching in terms of intra-level cooperation has been performed. Korupolu et al. [8] investigated this problem in it's general form, allowing any number of levels in the hierarchy. In their model caches are all leaf nodes and two caches can cooperate each other incurring a cost that is equal to the total costs in the path that unites them. They presented an exact algorithm based on reduction to a min-cost flow problem of computational complexity quadratic to the number of different content objects, which generalizes the results in [13] (in [13] authors studied only 2-level hierarchies). The high complexity of the above algorithm lead authors to provide a constant factor distributed approximation algorithm that is at most 13.93 times from optimal. This algorithm is two stage as in the first stage it computes a content pseudoplacement in caches (i.e. a content placement that does not respect capacity constraints of caches) and in the second stage it transforms it to a valid placement.

Laoutaris et al. showed that the problem can be expressed as a game in which players are caches, strategy space of each cache is the decision of placing an object or not and utility function of each cache is the difference of average access cost of requests of clients of that cache under a cache placement P from the average access cost if no content placement in caching was used. They proposed the following algorithm:
TSLS Algorithm

1) Each cache i compute it's initial placement P_{i0}, by placing the most popular objects according to its clients without violating its capacity constraints.
2) Caches are sorted in an arbitrary order
3) According to the previous order each cache i improves its content placement by picking a local placement that is a best response to the current global placement at this step

A Nash Equilibrium point is achieved, in which no cache has incentive to change its content placement and, as authors proved, a better global performance is achieved than in the case of no cooperation.

Borst et al. [10] studied distributed low-complexity content placement algorithms in both types of cooperation in two level hierarchical topologies, where requests for objects are generated in leaf caches. For the intra-level cache cooperation scenario they proposed the following algorithm:
Local-Greedy-Gen Algorithm (LGG)

- Start with an initial content placement
- Iteratively:

- Select a node i and an object n. If object n is currently not stored at node i,then replace that by some object m that is currently stored at node i, if and only if that increases the global utility

LGG algorithm needs as input an initial content placement of caches which can be computed in various ways. In [10] authors studied the cases that as initial placement were chosen the most popular objects in each cache (Full replication placement) or as initial placement was decided each of the $\sum_{\text{cache i}} |C_i|$ most popular objects according to total popularity to be stored one time in a cache (No replication placement). Figure 10 shows the convergence of Local greedy gen algorithm to the optimal content placement when it gets as initial content placement: full replication, no replication and random placement. Besides that authors in [10] reduced the discussed problem to a two knapsack problem and proved structural properties of the solution to the problem. They also showed that LGG algorithm achieves a 2-approximation ratio for the metric of maximizing cost savings.

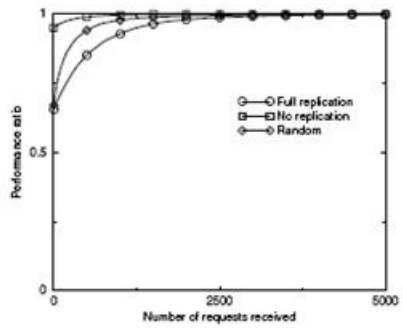

Figure 10

Inter-level cache cooperation PRP in a two level hierarchy is shown in figure 11.

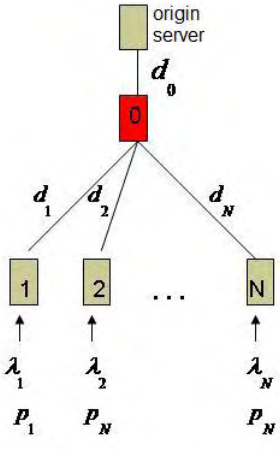

Figure 11

Observe that, unlike intra-level case, there is additionally a parent cache-node C_0. Leaf caches can not take objects one from the other and the only cooperation that is allowed is between parent node and each of the leaf caches. Similar to the intra-level case this problem can be equivalently seen as the problem of maximizing average cost saving instead of minimizing average access cost of client requests. The problem formulation is the following:

$\max_x \sum_{object\ j} \sum_{cache\ i} (\lambda_i p_{ij} |O_j| x_{ij} (d_0 + d_i) + \lambda_i p_{ij} |O_j| x_{i0j} d_0)$

subject to:

$$\sum_{object\ j} x_{ij}|O_j| \leq |C_i|, \forall \text{ cache i}$$
$$x_{i0j} \leq x_{0j}, \forall \text{ leaf cache i, } \forall \text{ object j}$$
$$x_{ij} + x_{i0j} \leq 1, \forall \text{ leaf cache i, } \forall \text{ object j}$$
$$x_{ij} \in \{0,1\}, \forall \text{ cache i, } \forall \text{ object j}$$
$$x_{i0j} \in \{0,1\}, \forall \text{ leaf cache i, } \forall \text{ object j}$$

where,

$$x_{i0j} = \begin{cases} 1 & \text{if leaf cache i takes object j from parent cache 0} \\ 0 & \text{else} \end{cases}$$

Authors in [10] proposed a heuristic greedy algorithm for dealing with inter-level cache cooperation problem. According to this algorithm each leaf cache stores the most popular objects of it's clients without violating it's capacity constraints. Then parent cache stores the most popular objects according to the requests of all the clients that have not been stored in any of the leaf caches without violating it's capacity constraints. They proved that greedy achieves at least a fraction $\frac{(N-1)d_{min}+Nd_0}{(N-1)d_{min}+(2N-1)d_0} \geq \frac{N}{2N+1}$ of the maximum achievable bandwidth savings, where $d_{min} = min_{i=1,..,N} d_i$.

Now we mention the related work performed in cooperative caching area for studying simultaneously more than one control problems, involving content placement problem. Laoutaris et al. [14] consider the joint problem of optimal location of the objects together with the capacity dimensioning of the proxies. Another study by Laoutaris et al. [15] addresses the storage capacity allocation problem for CDNs, which takes into account the optimal location of the proxies, the capacity of each proxy and the objects that should be placed in each proxy. Almeida et al. [16] considered the problem of jointly

routing requests and placing proxy servers in streaming CDNs. Authors presented an optimization model for the problem and proposed a number of heuristics for its solution in an attempt to minimize the total server and network delivery cost. More recently, Nguyen et al. [17] considered the problem of provisioning the so-called overlay distribution networks, which includes proxy server placement, request routing and object replication. These authors proposed an integer linear programming formulation along with a heuristic solution algorithm based on Lagrangean relaxation. In [18] Bektas et al. considers a design problem arising in CDNs, and simultaneously solves three problems: (i) the number and the location of the proxy servers to be used in the CDN among a given set of potential sites (proxy server placement), (ii) the objects to be located in each proxy server (object placement), and (iii) the assignment of each client to a proxy server (request routing). In [11] authors only considered the object placement and request routing subproblems.

Study of cooperative caching in P2P systems was initially performed in [19], where authors show that depending on the distance between two neighbor peers it may be beneficial or not to cooperatively perform content placement. Other investigations of content placement in P2P video on demand systems were performed by Suh et al. in [20], where performance analysis of both queuing and loss models are considered. However, optimal content placement in P2P video on demand systems with respect to the content popularity was first performed in [21].

III. SYSTEM MODEL

We now introduce our mathematical model and related notations. For convenience we use the same notations as in table 1 presented in previous section. Additionally, we specify that all objects are of the same size, normalized to 1. Topology is an hierarchy, with N leaf caches (indexed from 1 to N), where inter level cooperation is allowed, i.e a request generated at cache i follows the direct root to the origin server until it reaches a cache that has stored the requested object. Thus a request routed in a cache can not be routed to another cache in the same level of the hierarchy (intra-level cooperation is not permitted). External requests are generated only in leaf caches and not in internal cache-nodes of the hierarchy. The above assumptions are used throughout the paper without any violation.

In sections IV-V we make additionally the following system model assumptions: topology is a two level hierarchy where inter-level cooperation is allowed, with N leaf caches and a parent cache indexed to 0. Origin server is indexed as -1. We simplify the model by defining that the cost for transmitting a piece of unit over a link between parent cache and a leaf cache is 0 i.e. $d_{i0} = 0 \; \forall \; i \in C$ and the cost for traversing a link between origin server and parent cache and is 1 i.e. $d_{0,-1} = 1$. Thus we are only interested in minimizing the average rate of requests reaching origin server. We call the probability of accessing origin server probability of missing of the system, denoted by M. The problem of minimizing the probability of missing in the described system may then be formulated as follows: $minM = min\frac{\sum_{i \in C} \lambda_i M_i}{\sum_{i \in C} \lambda_i}$,where $M_i = 1 - \sum_{j \in C_i \cup C_0} p_{ij}$ is the probability of missing of the requests generated in cache i. The optimization variables are the content of caches. An equivalent way to express the problem is maximizing the probability a stream to find the requested content in any of the caches of the topology that has access and not to reach origin server (probability of hit). An important observation is that in optimal content placement the sets of the objects cached in each of the leaf caches must be disjoint to the set of objects stored in parent cache, as in any other case we could swap the common object with an unused increasing probability of hit. Based on that observation the formulation of the problem becomes:

$\max_x \sum_{object\ j} \sum_{cache\ i} (\lambda_i p_{ij} x_{ij})$

subject to:

$$\sum_{object\ j} x_{ij} \leq |C_i|, \forall \text{ cache i}$$

$$x_{ij} + x_{0j} \leq 1, \forall \text{ leaf cache i, } \forall \text{ object j}$$

$$x_{ij} \in \{0, 1\}, \forall \text{ cache i, } \forall \text{ object j}$$

Figure 12 shows the discussed system model:

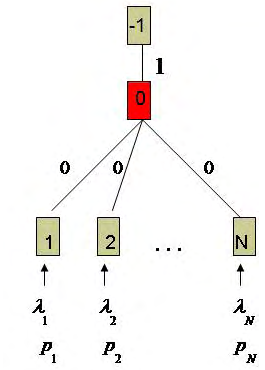

Figure 12

In section VI we make the same additional system model assumptions as in sections IV-V but now we allow the variables d_{i0} and $d_{0,-1}$ to take any value. This value will represent the distance/cost between each pair of connected nodes. Thus, we are interested in minimizing the average access cost incurred by the routing of requests for content, or equivalently maximizing the average cost savings. The problem formulation is the following:

$\max_x \sum_{object\ j} \sum_{cache\ i} (\lambda_i p_{ij} x_{ij} (d_{0,-1} + d_{i0}) + \lambda_i p_{ij} x_{i0j} d_{0,-1})$

subject to:

$$\sum_{object\ j} x_{ij} \leq |C_i|, \forall \text{ cache i}$$
$$x_{i0j} \leq x_{0j}, \forall \text{ leaf cache i, } \forall \text{ object j}$$
$$x_{ij} + x_{i0j} \leq 1, \forall \text{ leaf cache i, } \forall \text{ object j}$$
$$x_{ij} \in \{0,1\}, \forall \text{ cache i, } \forall \text{ object j}$$
$$x_{i0j} \in \{0,1\}, \forall \text{ leaf cache i, } \forall \text{ object j}$$

where,

$$x_{i0j} = \begin{cases} 1 & \text{if leaf cache i takes object j from parent cache 0} \\ 0 & \text{else} \end{cases}$$

Observe that now there is no restriction the object set stored in parent cache and the content stored in each leaf cache to be disjointed sets. Figure 13 shows the discussed system model:

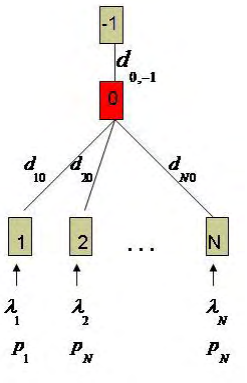

Figure 13

In section VII we make the same additional system model assumptions as in section VI but now we allow any number of levels of the hierarchical topology instead of only two.

IV. A COMMONLY USED AND TWO ATOMIC CACHES SCENARIO

In this section we deal with the simple case of inter level cache cooperation in a two-level hierarchy consisted of N=2 leaf caches and one parent cache. Request generated only in leaf caches, and a newly request access it's private cache and a commonly used (parent) cache in order to be satisfied. A miss is performed if a requested object is not placed nor in the leaf cache that is generated nor in the commonly used cache. Figure 14 shows the discussed cache topology.

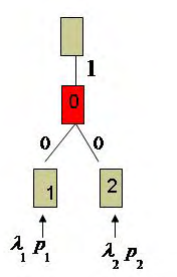

Figure 14

The probability of missing of requests for objects is $M_1 = 1 - \sum_{j \in C_1 \cup C_0} p_{1j}$ for requests generated at cache 1 and $M_2 = 1 - \sum_{j \in C_2 \cup C_0} p_{2j}$ for requests generated at cache 2. The optimization problem is to find the configuration of the three caches that minimizes the probability of missing of the system:

$minM = min \frac{\lambda_1 M_1 + \lambda_2 M_2}{\lambda_1 + \lambda_2}$ (1)

In this section we reduce the content placement problem (1) to the problem of finding the maximum weight perfect matching in a bipartite graph. We construct an instance G of the maximum weight perfect matching in bipartite graph problem that it's solution yields a minimum probability of missing content placement.

IV.a The reduction. The instance G is constructed as follows. The vertex set consists of the following: (i) two vertices j,j' for every object j of object set O, (ii) a vertex C_1^p for every position p of the cache C_1 that needs to be filled in, (iii) a vertex C_2^l for every position l of the cache C_2 that needs to be filled in, (iv) a vertex d_x for every object of the object set that will not be placed in any of the C_0 or C_1 where $x = 1, 2, ..., |O| - |C_1| - |C_0|$ and (v) a vertex $d_y^{'}$ for every object of the object set that will not be placed in any of the C_0 or C_2, where $y = 1, 2, ..., |O| - |C_2| - |C_0|$. The edge set consists of the following types of edges: (i) for each object j and j' there is an edge (j,j') with weight $\lambda_1 p_{1j} + \lambda_2 p_{2j}$, (ii) for each vertex j and vertex C_1^p there is an edge (j,C_1^p) with weight $\lambda_1 p_{1j}$, (iii) for each vertex j' and vertex C_2^l there is an edge (j',C_2^l) with weight $\lambda_2 p_{2j'}$, (iv) for each vertex j and vertex d_x there is an edge (j,d_x) with zero weight,(iv) for each vertex j' and vertex d'_y there is an edge (j',d'_y) with zero weight. The described graph is shown in figure 15.

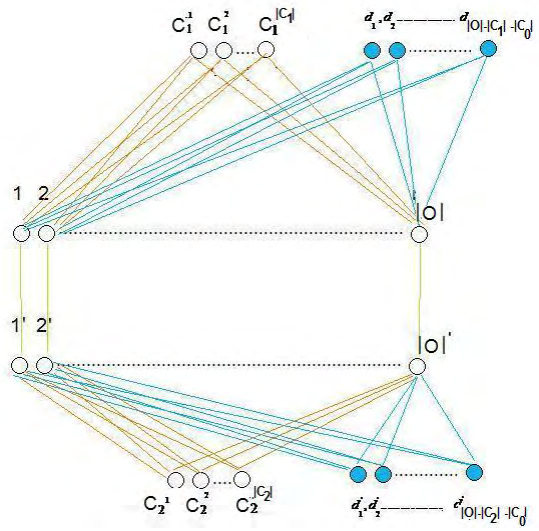

Figure 15

Graph G is bipartite as vertices are separated in two parts, one that contains vertices named as j, C_2^l, d'_y and the other part that contains vertices named as j', C_1^p, d_x, where $j = 1, ..., |O|$, $j' = 1', ..., |O|'$, $p = 1, ..., |C_1|$, $l = 1, ..., |C_2|$, $x = 1, 2, ..., |O| - |C_1| - |C_0|$, $y = 1, 2, ..., |O| - |C_2| - |C_0|$. There are no links with both their endpoints in the same part of the graph. The graph is of size nXn where $n = 2|O| - |C_0|$.

The proposed Algorithm named as MWPMCP (Maximum weight perfect matching Content Placement) is the following:

1. Create the bipartite graph G according to the previous directions
2. Find a maximum weight perfect matching m in the bipartite graph G, of weight W
3. Calculate probability of missing of the system $M = 1 - \frac{W}{\lambda_1 + \lambda_2}$
4. For every edge (e1,e2) that is selected in m
   a. Assign object j to cache C_0, if (e1,e2)=(j,j')
   b. Assign object j to cache C_1, if (e1,e2)=(j,C_1^p)
   c. Assign object j to cache C_2, if (e1,e2)=(j',C_2^l)

IV.b Proof of optimality. We now prove that the preceding reduction yields the optimal solution according to algorithm MWPMCP, by proving the next three theorems.

Theorem 1: Every perfect matching in the graph G corresponds to an assignment of objects to the caches such that the number of assigned objects in each cache is equal to the size of the cache.

Proof

Computing one of the possible perfect matching m in the graph G, and then assigning objects to caches according to step 4 of the MWPMCP Algorithm specifies that the number of assigned objects in each cache is equal to the size of the cache. This can be proved as follows:

Suppose that a perfect matching in G corresponds to an assignment to cache C_0 of

$|C_0|$+n objects, where $n > 0$. Then $|C_0|$+n pairs of vertices (j,j'), where $j \in O$ were selected during the calculation of perfect matching in the corresponding bipartite graph G. By the structure of the graph G, the selection of edges (j,C_1^p) and of edges (j,d_x) will surely result that the degree of at least one vertex $j \in O$ will be more than one, in order the assignment to be perfect. Thus, this assignment is no more a matching, which contradicts the hypothesis. So, every perfect matching assigns to cache C_0 at most $|C_0|$ objects.

Now, suppose that a perfect matching corresponds to an assignment to cache C_0 of $|C_0|$-n objects, where $n > 0$. Then $|C_0|$-n pairs of vertices (j,j'), where $j \in O$ were selected during the calculation of perfect matching in the corresponding bipartite graph G. By the structure of the graph G, the selection of edges (j,C_1^p) and of edges (j,d_x) will surely result that at least one vertex $j \in O$ will not be matched, in order the assignment to be a matching. Thus, this assignment is no more a matching, which contradicts the hypothesis. So, every perfect matching assigns to cache C_0 at least $|C_0|$ objects.

So, every perfect matching assigns to cache C_0 exactly $|C_0|$ objects.

Similarly, we can conclude that every perfect matching assigns to cache C_1 exactly $|C_1|$ objects and to cache C_2 exactly $|C_2|$ objects.

Theorem 2: Every perfect matching in the graph G corresponds to an assignment of objects to the caches C_0, C_1, C_2 such that $C_0 \cap C_1 = \emptyset$ and $C_0 \cap C_2 = \emptyset$.

Proof

Computing one of the possible perfect matching m in the graph G, and then assigning objects to caches according to step 4 of the MWPMCP Algorithm specifies that the set of the assigned objects the common cache is disjoined from each of the sets of the assigned objects in each atomic cache. This can be proved as follows:

Assume that $C_0 \cap C_1 \neq \emptyset$ and name $j \in O$ an object in $C_0 \cap C_1$. Then, according to the definition of step 4 of the MWPMCP algorithm, both links (j,j') and (j,C_1^p), where $p \in (1, ..., |C_1|)$, were selected during the calculation of perfect matching in the corresponding bipartite graph G. Then, the degree of node j in the graph that came from G, by keeping all the vertex set of G and include in edge set all the edges of G that are selected by perfect matching procedure, must be at least two. That contradicts the statement that the links (j,j') and (j,C_1^p) were selected in the perfect matching of graph G, as the degree of every vertex that belongs to the matching must be exactly one. So, $C_0 \cap C_1 = \emptyset$.
Similar we prove that $C_0 \cap C_2 = \emptyset$.

Theorem 3: MWPMCP Algorithm calculates the minimum probability of missing of the system

Proof

Theorems 1 and 2 showed that every perfect matching corresponds to an assignment of the objects to caches. By definition, the maximum weight perfect matching finds the maximum value W of the following expression between all the other choices of perfect matching:
$W = max \sum_{j \in C_1} \lambda_1 p_{1j} + \sum_{j \in C_2} \lambda_2 p_{2j} + \sum_{j \in C_0} \lambda_1 p_{1j} + \lambda_2 p_{2j}$. So, It holds:
$1 - \frac{W}{\lambda_1 + \lambda_2}$ =
= $\min 1 - \frac{\sum_{j \in C_1} \lambda_1 p_{1j} + \sum_{j \in C_2} \lambda_2 p_{2j} + \sum_{j \in C_0} \lambda_1 p_{1j} + \lambda_2 p_{2j}}{\lambda_1 + \lambda_2}$

Because of theorem 2, it holds:

$$1 - \frac{W}{\lambda_1+\lambda_2} = \min 1 - \frac{\sum_{i\in C_1\cup C_0}\lambda_1 p_{1j} + \sum_{j\in C_2\cup C_0}\lambda_2 p_{2j}}{\lambda_1+\lambda_2}$$

$$= \min \frac{\lambda_1+\lambda_2-(\sum_{j\in C_1\cup C_0}\lambda_1 p_{1j} + \sum_{j\in C_2\cup C_0}\lambda_2 p_{2j})}{\lambda_1+\lambda_2}$$

$$= \min \frac{\lambda_1(1-\sum_{j\in C_1\cup C_0}\lambda_1 p_{1j}) + \lambda_2(1-\sum_{j\in C_2\cup C_0}\lambda_2 p_{2j})}{\lambda_1+\lambda_2}$$

$$= \min \frac{\lambda_1 M_1+\lambda_2 M_2}{\lambda_1+\lambda_2}$$

$$= \min M$$

IV.c Computational complexity analysis. The computational complexity of MW-PMCP algorithm comes mainly to the calculation of the max weight perfect matching in the bipartite graph G, of dimension $2|O| - |C_0|$. It is known that algorithm proposed by Kuhn (also known as hungarian method), finds the max weight perfect matching with complexity cubic to the dimension of the graph, resulting complexity O($(2|O| - C_0)^3$). The rest of the calculations performed by algorithm is less significant than the cubic factor.

V. A CLUSTER OF CACHES SCENARIO

We now focus on an extension of the problem described above: the case that the number of leaf caches is greater or equal than two ($N \geq 2$) and there is again one parent cache. Requests generated only in leaf caches, and a newly request access the leaf cache that is generated and the parent cache in order to be satisfied. A miss is performed if a requested object is not placed nor in the leaf cache that is generated nor in the parent cache.

To the best of our knowledge MWPMCP algorithm can not be extended to handle this generalized case. Thus, in this section we present an optimal algorithm based on different philosophy that solves the problem of content placement that minimizes the probability of missing of the generalized system: $minM = min\frac{\lambda_1 M_1 + ... + \lambda_N M_N}{\lambda_1 + ... + \lambda_N}$, where Mi is the probability of missing of requests generated in leaf cache i, for i=1,,N and λ_i is the rate of requests of stream i. We name this algorithm Dynamic, because it dynamically fills in each of the available positions of parent cache C_0 by placing objects in it one by one. Furthermore, we present low complexity heuristic algorithms that approximate the optimal solution.

V.a Dynamic Algorithm

The algorithm is based on the following two properties:

Property 1:

Given the placement of C_0 the best content placement to leaf caches is trivial to find (i.e. place in leaf cache C_i the most popular objects according to pi, not included in C_0).

Property 2:

The optimal content placement of the parent cache when it is of size K is a subset of the optimal content placement of the parent cache when it is of size K+1 and the number and sizes of the leaf caches, the request patterns of users and the object set remain the same in both cases.

Property 2 will be proved strictly by theorem 4. Before describing the structure of the algorithm we first present the function exhaustive(n_0,n_1,...,n_N,J) which return the optimal content placement of common cache when $|C_0| = n_0$, $|C_1| = n_1$,.., $|C_N| = n_N$ and $O = J$ (assuming that user request patterns are fixed and known). This function calculates its result by exhaustively, comparing the probability of missing of all the different content placements in each of the caches. Dynamic algorithm iteratively calls function exhaustive by always passing as parameter $n_0 = 1$,$n_1 = |C_1|$,.., $n_N = |C_N|$ i.e. finding the optimal content placement when size of common cache is 1 and the other sizes are the same to the ones of the true topology. The only difference in callings of function exhaustive() by dynamic algorithm is the passing value on parameter J. We define as $C_0[p]$ the object stored in position p of parent cache C_0. Analytically, the algorithm works as follows:

1) $J = O$
2) For each position $p = 1, .., |C_0|$ of the common cache:
   a) $C_0[p] = exhaustive(1, |C_1|, ..., |C_N|, J)$
   b) $J = J \setminus C_0[p]$

We reduced the non-polynomial solvable problem of content placement in a 2-level hierarchy of a parent cache of size $|C_0|$ to a sequence of $|C_0|$ polynomial solvable problems of content placement in a 2-level hierarchy of parent cache of size 1. However these problems can not be solved in parallel as the output of the p^{th} problem is input for the $(p+1)^{th}$ problem. Figure 16 illustrates the split of the initial placement problem.

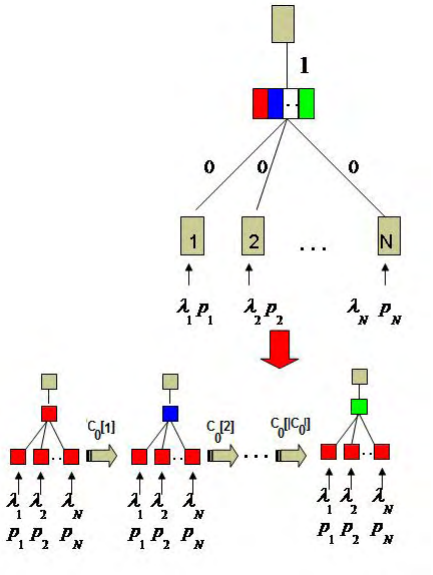

Figure 16

We define as $C_0^*(n, O)$ the subset of size n of object set O, that corresponds to the optimal content placement of the parent cache, when $|C_0| = n$, and the sizes of the leaf caches are constant $|C_1|, .., |C_N|$.

Theorem 4: $C_0^*(n-1, O)$ is a subset of $C_0^*(n, O)$.

Proof

The proof of the theorem is described in the Appendix.

Theorem 5 proves the optimality of the proposed Algorithm.

Theorem 5:

$C_0^*(n, O) = C_0^*(n-1, O) \cup C_0^*(1, O \setminus C_0^*(n-1, O))$.

Proof

By theorem 4 and the definition of $C_0^*(1, O \setminus C_0^*(n-1, O))$.

Observe that execution run time of function exhaustive is polynomial when it is called with size of parent cache equal to 1, and non-polynomial in the general form that is called with size of parent node $n_0 > 1$. For the case that passing parameter $n_0 = 1$, the complexity of execution of exhaustive() is $N|O|^2$, as it exhaustively calculates the missing achieved by each of the $|O|$ objects that can be placed in parent cache of size 1, placing at the same time the most popular objects in each of the N leaf caches, by searching it's vector p_i of size $|O|$, for finding the $|C_i|$ most popular objects. Assuming that p_i vectors are initially sorted, the cost of this search of a vector is at most equal to

it's size, $|O|$. Thus, The computational complexity of the proposed algorithm, Dynamic, is $O(N|O|log|O| + N|C_0||O|^2)$, where the first term of the sum is for the initial sorting of the object probability vectors and the second term is for the $|C_0|$ calls of function exhaustive() with parameter $n_0 = 1$.

V.b Swapping Algorithm

We present the following simple heuristic swapping algorithm:
Algorithm Swapping:
start with a random content placement in all the caches such that no double entries in the same cache exist
do
 for each object j
  for each cache i
   find the replacement of object j with an object already in
   a cache that gives the largest decrement in probability
   of missing of the system
while probability of missing is lower after replacement

V.c Greedy Algorithm

We present the following simple greedy algorithm:
Fill in each leaf cache C_i with the most weighted objects according to it's popularity vector p_i (first-tier objects).
Fill in the parent cache C_0 with the most weighted objects according to the sum of popularity of all the requests $p_1 + ... + p_N$ that were not included in any of the leaf caches in the previous step (second-tier objects).

V.d A lower bound of performance

Probability of missing of the above system is lower bounded by the amount: $M_{LB} = 1 - \frac{\sum_{i \in C} \lambda_i \sum_{j=1}^{|C_0|+|C_i|} q_{ij}}{\sum_{i \in C} \lambda_i}$,where q_{ij} is the j^{th} largest probability among the values of the probability vector: $p_i = [p_{i1}, ..., p_{i|O|}]$. In simple words, M_{LB} represents the hypothetical scenario that there are N different parent caches instead of only 1 (all of them with the same size to the size of real parent cache), such that requests generated at each leaf cache has access at exactly one of these parent caches, and requests of two or more different leaf caches can not access the same parent cache.

VI. Extension to general cost model

Until now we concentrated on minimizing probability of missing of the system in 2-level hierarchical cache topologies in which inter level cooperation is allowed. This can be seen as the special case of minimizing average access cost of traversing of requests when cost of using each link between a leaf cache and parent cache is 0 and cost of using link between parent cache and origin server is 1. In this section we generalize some of the ideas presented above to deal with arbitrary link costs.

MWPMCP algorithm can not be extended to handle the case of arbitrary link costs in a topology of two leaf caches, as it depends on the idea that the content that is stored in parent cache must be disjoint with the content stored in each of the leaf caches. This principle was right in the case of minimizing probability of missing. However it does not hold for arbitrary link costs. (Property 1 does not hold now).

From the other hand dynamic algorithm can be extended to handle the case of arbitrary link costs in a topology of N leaf caches. We only need to change the operation of function exhaustive in order to allow the algorithm to place an object in a leaf cache that already is placed in parent cache. We change the set of parameters of function exhaustive to be (n_0,n_1,...,n_N,J,$already[]$) which return the optimal content placement of common cache when $|C_0| = n_0$, $|C_1| = n_1$,.., $|C_N| = n_N$ and $O = J$, assuming that objects in matrix already[] are stored in parent cache without consuming any of the n_0 free positions of C_0, and so can satisfy requests for that that reach parent cache. Thus, calculation of cost saving of placing an object in a cache depends on the objects already placed in parent cache. This function calculates its result by exhaustively, comparing the probability of missing of all the different content placements in each of the caches.

We call the generalized form of dynamic algorithm that handle link costs as CostDynamic. Analytically, the algorithm works as follows:

1) $C_0[]$ =empty set

2) For each position $p = 1, .., |C_0|$ of the common cache:
   a) $C_0[p] = exhaustive(1, |C_1|, ..., |C_N|, O, C_0[])$

The passing value of parameter $C_0[]$ in the call of exhaustive function by CostDynamic algorithm is the set of objects already placed in parent cache until the execution of p step of the loop. Observe that according to the above definition of exhaustive() there is no chance an object already placed in parent cache to be placed again by CostDynamic algorithm. The difference to Dynamic algorithm is that now algorithm allows for additional placements in leaf caches of an object that already being placed in common cache at previous step. The complexity of CostDynamic algorithm remains polynomial due to the same arguments discussed in complexity analysis of Dynamic Algorithm.

VII. Inter-level cooperative content placement in multi level cache hierarchies

CostDynamic algorithm can be extended to find optimal content placement of caches in an hierarchy of arbitrary number of levels $H \geq 2$, where inter level cooperation is allowed. We call this algorithm RecAlgorithm as it solves the problem in a recursive manner starting from the root of the hierarchy. The idea is that each internal cache-node in the hierarchy represents a content placement problem involving it and it's descendants. The initial problem is represented by the root cache-node. The internal nodes in level 1 represent 2-level hierarchy content placement problems and can be solved by application of algorithm CostDynamic. Problems represented by cache nodes in higher levels can be solved, using the same philosophy, by placing objects in them one by one, picked after exhaustive comparison of the cost saving of all object choices, which can be calculated using the optimal content placement in the children of them computed in a recursive manner.

Assuming that CostDynmic() and RecDynamic() functions return the cost saving of the optimal content placement that calculated in the hierarchy specified by their input parameters, the algorithm works as follows:

Initially we call this method RecDynamic() with $current = root$ and $level = H$.

RecDynamic(current, level)
  if current is on level 1
    Return CostDynamic(current)
  Else if current is on $level > 1$
    max=0
    For each position p of cache current
      max(p)=0
      For each object j
        max(p,j)=0
        For each child Ci of current
          $max(p,j) = max(p,j) + RecDynamic(Ci, level-1)$
          if $max(p,j) > max(p)$
            $max(p) = max(p,j)$
        $max = max + max(p)$
      $return\ max$

where max(p,j) is the cost saving incurred by placing object j in position p of cache current in combination to the optimal content placement of children nodes of current. max(p) is the largest cost saving achieved by placing an object in position p of cache current and max is the total cost saving incurred by all the content placement in cache current and it's descendants.

Figure 17 shows a content placement problem in a four-level hierarchy. In order to solve problem P0, RecDynamic solves the P11 and P12 problems for each of the possible object placed in each of the positions of C_0 cache. Similarly, P11 problem requires repeat solution of P21 and P22 and P12 requires repeat solution of P23 and P24 problems.

P21,P22,P23,P24 problems are solved each time using CostDynamic Algorithm.

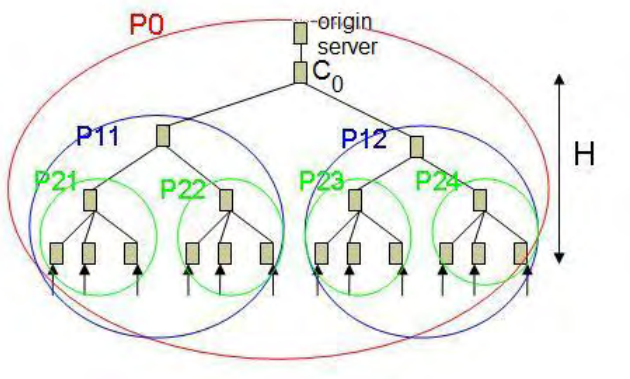

Figure 17

The complexity of RecDynamic Algorithm is polynomial: $|O|^{H+1}|C_0| \prod_{l=0}^{H-1} nodes(l)$, where nodes(l) is the number of cache-nodes in H-l level.

VIII. JOINT CONTENT PLACEMENT AND CLIENT ASSIGNMENT PROBLEM

Until now we assumed that stream of requests are a priori assigned to N leaf caches, described by popularity vectors p_i and request rates λ_i, $\forall$ leaf cache i. However, assuming now that the clients that generate these streams are mobile, and because of this ability can choose to be assigned to any leaf cache, a new problem of joint client assignment to leaf caches and content placement in caches of the hierarchy arises. We assume that each client can be assigned to exactly one leaf cache and this assignment is static, i.e. a client assigned to a leaf cache i can not later move and be assigned to another cache i'. For simplicity, we assume that the client assignments are of zero cost. The stream assignment should be performed jointly with the content placement problem in order to optimally minimize the average access cost of traversing of requests for objects in links of hierarchy.

In order to handle the client assignment problem of M clients, we define as S the set of clients indexed as $\{1, ..., M\}$. Additionally we change the definition of vectors λ and p in order to refer to clients instead of leaf caches, i.e. $\lambda_s =$ the rate that client s generates requests for content and $p_{sj} =$ the probability that a request generated by client s is for object j,for $s = 1, .., M$, j=1,..,$|O|$. Additionally, we change the definition of vector x and introduce a new vector z, defined as follows:

$$x_{si} = \begin{cases} 1 & \text{if client s is assigned to cache i} \\ 0 & \text{else} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if object j is stored in cache i} \\ 0 & \text{else} \end{cases}$$

$$\forall s = 1, .., M,\ \forall i = 1, .., N,\ \forall j = 1, .., |O|$$

Below we present methodology for the solution of the joint client assignment and content placement problem in two-level hierarchies of caches where no cooperation is allowed between caches or inter level cooperation between leaf caches and parent cache is allowed in order to respond to requests for content.

VIII.a Non-cooperation cache model

Figure 18 shows the system topology. Observe the absence of a parent cache-node between origin server and leaf caches as it is not allowed inter-level cache cooperation. Additionally it is not allowed for a leaf cache to take a requested object from another leaf cache as it is not allowed intra-level cache cooperation. As a result requests for content generated in a leaf cache i can be satisfied without cost by cache i, or incurring a cost of d_{i0} from origin server.

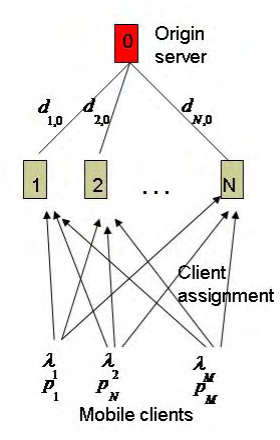

Figure 18

Based on the above discussion a formulation of the joint problem is the following:

(P) $\min_{x,z} \sum_{client\ s} \sum_{cache\ i} \sum_{object\ j} (\lambda_s p_{sj} |O_j| x_{si} d_{i,0} - \lambda_s p_{sj} |O_j| x_{si} z_{ij} d_{i,0})$

subject to:

$$\sum_{cache\ i} x_{si} = 1, \forall \text{ client s } (1)$$

$$\sum_{object\ j} |O_j| z_{ij} \leq |C_i|, \forall \text{ cache i } (2)$$

$$x_{si} \geq 0, \forall \text{ cache i, } \forall \text{ client s } (3)$$

$$z_{ij} \in \{0, 1\}, \forall \text{ cache i, } \forall \text{ object j } (4)$$

Based on the methodology described in [11] for solving the joint problem of content placement and routing of requests satisfying QoS constraints in an arbitrary cache topology, we describe a subgradient method that finds an exact solution of (P).

Observe that problem (P) is non-linear because of the product term $x_{si} z_{ij}$ in the objective function, which makes it difficult to be solved. A linearization procedure is needed. We choose a linearization procedure based on that proposed by Glover [22]. Particularly, we choose as linearizing variable the $v_{ij} = z_{ij} \sum_{clients} |O_j| \lambda_s p_{si} d_{i0} x_{si}$, incurring the below additional constraints:

$v_{ij} \leq M z_{ij}$, $\forall cachei$, $\forall objectj$ (5)

where $M = \sum_{clients} |O_j| \lambda_s p_{si} d_{i0}$

$v_{ij} \leq \sum_{clients} |O_j| \lambda_s p_{si} d_{i0} x_{si}$ (6)

After (P) is linearized, we relax constraint (5), by introducing a lagrange multiplier γ_{ij}, for each of the constraints in (5) and adding the product of multiplier and related constraint term in objective function. Thus, a new problem (RP) is formulated. Observe that the solution of (RP) is a lower bound to the solution of (P). Problem (RP) decomposes to the following two subproblems (RP1) and (RP2), which can be solved individually one from the other given that we know the values of lagrange multipliers γ.

(RP1) $\min_{v,x} \sum_{client\ s} \sum_{cache\ i} \sum_{object\ j} (\lambda_s p_{sj} |O_j| x_{si} d_{i,0}) + \sum_{cache\ i} \sum_{object\ j} (\gamma_{ij} - 1) v_{ij}$

subject to:

$$\sum_{cache\ i} x_{si} = 1, \forall \text{ client s}$$

$$v_{ij} - \sum_{stream\ s} \lambda_s p_{sj} |O_j| d_{i0} x_{si} \leq 0, \forall \text{ cache i}, \forall \text{ object j}$$

$$x_{si} \geq 0, \forall \text{ client s}, \forall \text{ cache i}$$

$$v_{ij} \geq 0, \forall \text{ cache i}, \forall \text{ object j}$$

(RP2) $\max_z M \sum_{cache\ i} \sum_{object\ j} (\gamma_{ij} z_{ij})$

subject to:

$$\sum_{object\ j} z_{ij} \leq |C_i|, \forall \text{ cache i}$$

$$z_{ij} \in \{0, 1\}, \forall \text{ cache i, } \forall \text{ object j}$$

Given that we know the values of lagrange multipliers γ, (RP1) can be solved simply by an assignment rule:

$$v_{ij} = \begin{cases} 0 & \text{if } \gamma_{ij} - 1 \geq 0 \\ \sum_{clients} (|O_j| \lambda_s p_{sj} d_{i0} x_{ij}) & \text{else} \end{cases}$$

$$x_{ip} = 1, p \in argmin_i \sum_j (|O_j| \lambda_s p_{sj} d_{i0}), \forall \text{client s}$$

As for subproblem (RP2), it is easy to show that it further decomposes into N unidimensional knapsack problems, one for each cache $i \in C$. It is known that each knapsack problem can be solved by dynamic programming in $O(|O||C_i|)$ for $i \in C$. Therefore, for

a given set of Lagrange multipliers γ subproblem (RP2) can be solved in $O(N|O||C_i|)$ time.

The subgradient method iteratively calculates a new set of lagrange multipliers γ, solves the problem (RP) by solving (RP1) and (RP2) and compares the solution of (RP) to the solution of (P) in order to check if the solutions are close enough to stop the procedure. We provide below an outline of the algorithm:

- Start with an initial multiplier γ^1
- $LB = -\infty$
- $UB = +\infty$
- t=1
- While $|UB - LB| > \varepsilon$
  - Find the solution of (RP) given γ^t, $v(RP(\gamma^t))$
  - If $v(RT(\gamma^t)) > LB$
    - $LB = v(RT(\gamma^t))$
  - $UB =$ the solution of (P) given the value of z calculated in $v(RP(\gamma^t))$
  - $g_{ij}^t = v_{ij} - Mz_{ij}$
  - $s^t = \lambda \frac{UB - v(RP(\gamma^t))}{|g^t|^2}$
  - $\gamma^{t+1} = max(0, \gamma^t + s^t g^t)$
  - $t = t + 1$

where λ and ε are constants.

VIII.b Inter level cache cooperation model

Now we study the above joint problem allowing inter level cooperation of leaf caches with a parent cache, which is located between leaf caches and origin server as in figure 19.

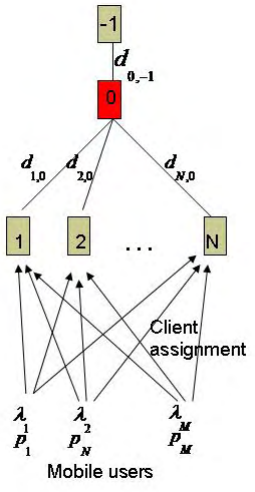

Figure 19

The subgradient method can not be applied as before, as the decomposition to subproblems is not as simple as before, because of the presence of the parent cache. Alternatively,

we present a heuristic algorithm based on the following idea: Clients with similar request patterns should be assigned to the same cache, in order to take from it the "most popular common objects". A clustering algorithm (k-means) can be used that assigns the clients into M clusters, one for each leaf cache. Given the assignment of clients calculated above, optimal content placement can be performed using CostDynamic algorithm described in previous sections. However, solving the two problems separately, implies that the solution to the joint problem will not be optimal but only a heuristic solution. A distance metric must be used by k-means algorithm in order to calculate the distance between a client and each cluster in order to decide at each step in which cluster each client must be assigned. This distance metric can be Euclidean distance or the sine metric [12]: $D(S1, S2) = sin(S1, S2) = 1 - \sqrt{1 - cos(S1, S2)^2}$, where:
$cos(S1, S2) = \frac{\sum_{objectj} \lambda_{S1} p_{S1j} \lambda_{S2} p_{S2j}}{\sqrt{\sum_{objectj} (\lambda_{S1} p_{S1j})^2} \sqrt{\sum_{objectj} (\lambda_{S2} p_{S2j})^2}}$, is the distance of clients S1, with rate λ_{S1} and popularity vector p_{S1j}, and S2, with rate λ_{S2} and popularity vector p_{S2j}.

We provide below an outline of the algorithm:

1) randomly select N clients as the centers of the clusters
2) for each unassigned client,assign it to the cluster whose center is the nearest to it. The distance between a client and the center is calculated by D function
3) Compute the means of the new clusters and assign clients to clusters with the new centers
4) If there is no change, exit. Otherwise go to step 2).

IX. Simulation results

In this section we present the results of the simulation performed for the evaluation of the proposed algorithms.

IX.a A commonly used and two atomic caches scenario

We assumed that the popularity of objects generated in every cache is a zipfian distribution, i.e. the frequency of any object is inversely proportional to its rank in the frequency table. Particularly, the popularity of the j^{th} object generated in cache i is: $p_{ij} = \frac{1/k_{i,j}^s}{\sum_{n=1}^{|}O|1/n^s}$, for i=1,2, for $j = 1, 2, ..., |O|$, where $k_{i,j}$ is the rank of the j^{th} object in the popularity vector p_i and s is the value of the exponent characterizing the distribution.

We simulated the described algorithms for constant number of objects $|O| = 100$, size of leaf caches $|C_1| = 5$, $|C_2| = 5$, rate of requests λ_1=λ_2=1 and popularity vectors p_1, p_2 picked according to zipfian distribution where $s = 0.8$ and $k_{i,j}$ were picked randomly.

Figure 20 shows the probability of missing of the system for increasing values of the size of parent cache $|C_0|$ (from 1 to 95) achieved by the algorithms we described. Blue square points are the values outputted by the optimal algorithm MWPMCP. Red cyclic points present the performance achieved by the heuristic swapping method and with the black-star line we denote the greedy algorithm. The top black line presents the worst case performance of the greedy algorithm as being defined in [10]. The cyan line is the lower bound to the probability of missing M_{LB} of the system as described in section V.

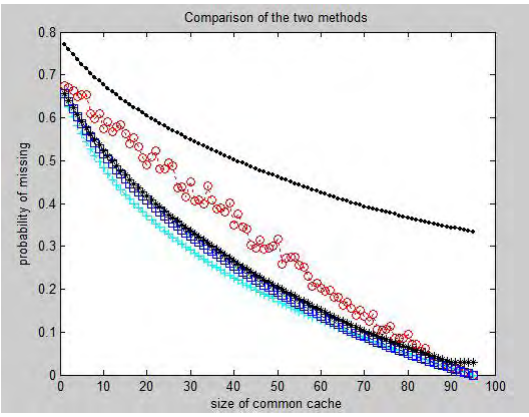

Figure 20

The difference of probability of missing of optimal and heuristic swapping algorithm due to the above simulation is shown in figure 21.

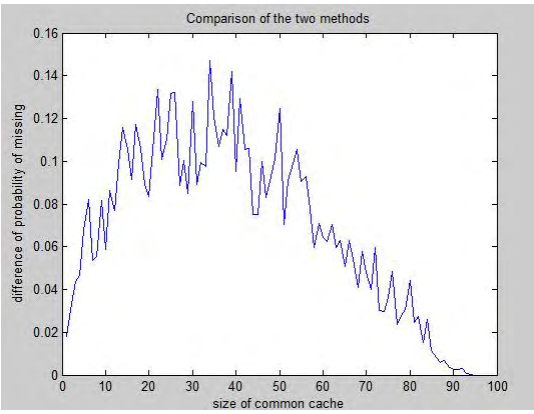

Figure 21

We note that the difference of probability of missing is proportional to the value of the expression $min(|O| - |C_1| - |C_2| - |C_0|, |C_1| + |C_2| + |C_0|)$. This is logical since in that cases the choice of the content of the common cache is a more crucial issue rather than in the rest cases.

IX.b A cluster of caches scenario

We simulated the algorithms described for a cluster topology for the case that the number of leaf caches is N=5, the number of object is constant and equal to $|O|$=100, the size of each one of the five leaf caches is set to be 5, the rate of requests of all the stream is equal and normalized to 1 and popularity vectors are picked according to zipfian distribution where skew parameter is set $s = 0.8$ for all the streams.

Figure 22 shows the probability of missing of the system for increasing values of the size of common cache $|C_0|$ (from 1 to 95) achieved by the algorithms we described. Blue square points are the values outputted by the optimal algorithm Dynamic. Red cyclic points present the performance achieved by the heuristic swapping method and with the black-star line we denote the greedy algorithm. The top black line presents the worst case performance of the greedy algorithm as being defined in [10]. The cyan line is the lower bound to the probability of missing M_{LB} of the system. Observe that the performance achieved by greedy algorithm does not get lower after the size of the parent cache becomes 75, that because the objects that are not picked by any leaf cache are less than the size of parent cache, and as a result some positions of parent cache remains empty. Simulation results verified that, as it is first described in [10], greedy algorithm is the most favorable choice between heuristics low complexity algorithms for inter level cooperation cache hierarchies (while swapping algorithm is a good choice for intra level cooperation cache hierarchies).

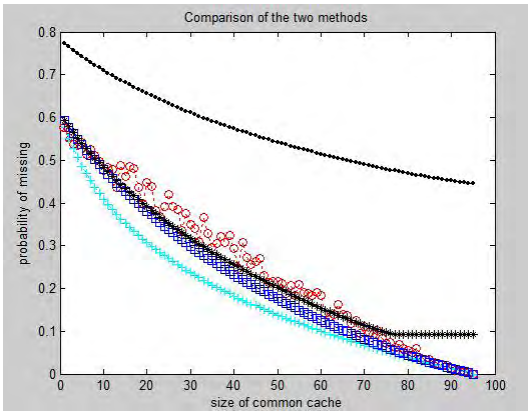

Figure 22

IX.c Effect of zipfian parameter in optimal solution

Figure 23 shows the average access cost of variation of size of parent cache for increasing values of zipfian parameter found by CostDynamic algorithm described in section VI. The topology used was a two level hierarchy with two leaf caches and a parent cache, while cooperation between leaf cache and parent was allowed. The cost of traversing a link between a leaf cache and parent cache was set to 1 and the cost for traversing link between parent cache and origin server was set to 2. The request rates was set to 1 and the popularity vectors were picked according to zipfian distribution of parameter s. The cache sizes of all cache was set to 5 and the size of object set to 100. The returned values coincide with the values found by exhaustive Algorithm, which evaluates one by one all the possible content placement in every cache,verifying the optimality of CostDynamic algorithm. We observe that as zipfian parameter s increases, the average access cost gets lower, as the selected objects that are stored in caches are more often requested than for low values of s.

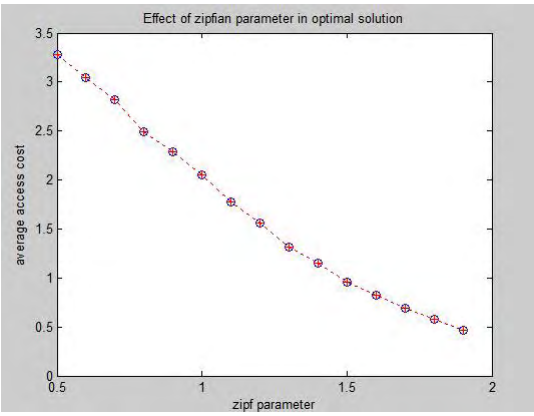

Figure 23

IX.d Effects of imperfect knowledge about input data

The above simulation results are based on the assumption that we have perfect knowledge of the number of requests generated from each node. In practice, we do not have perfect knowledge about these input, but only have rough estimates. In this section, we show how the inaccurate of request patterns affects the performance achieved by the heuristic content placement algorithms. In particular, we will show that the placement decision based on inaccurate information performed by the proposed heuristic algorithms is still useful. That is because, as we show, the difference in performance between the case of perfect information and imperfect information is small.

We distorted the popularity of request vectors for the parent cache and two leaf caches scenario, by a factor of μ. Particularly, for each node i and object j we chose uniformly at random a value between p_{ij}/μ and $p_{ij} * \mu$. Finally, we normalized these $|O|$ values earned for each node i, in order to sum to 1. We feed the previous incorrect inputs to the heuristic swapping and greedy algorithm for the same topology and the same numerical values for $|O|$, λ_1, λ_2, p_1 and p_2 as in section IX.a. Figure 24 shows the probability of missing of the system for increasing values of the size of parent cache $|C_0|$ (from 1 to 95) achieved by the algorithms we described. Blue square points are the values outputted by the optimal algorithm MWPMCP, red cyclic points present the performance achieved by the heuristic swapping method and with the black-star line we denote the greedy algorithm, when the information of the popularity of request vectors was perfect. Black-cross points present the performance achieved by the heuristic swapping method and with the cyan-dotted line we denote the greedy algorithm, when the information of the popularity of requests vectors was imperfect, by a factor $\mu = 3$.

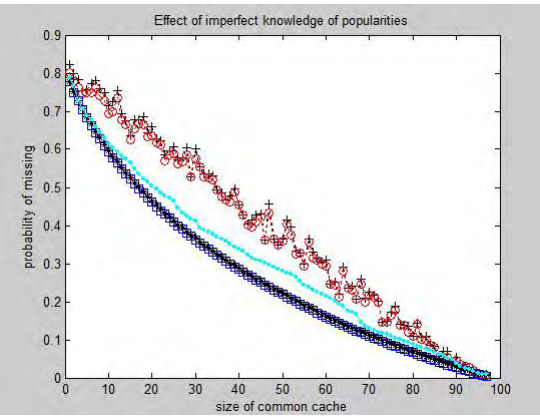

Figure 24

We note that heuristic swapping algorithm obtains performance that deviates slightly from that obtained using perfect knowledge. In contrast, greedy algorithm is not so robust to imperfect knowledge, as it deviates more than swapping algorithm in cases of imperfect information. However greedy's performance is for almost all values of common size better than performance of swapping algorithm, even when greedy algorithm operates under imperfect information and swapping under perfect information. In order to demonstrate

the deviation of the performance of these algorithms to the optimal algorithm's, figure 25 shows the the related performance of them for various values of size of common cache. By related performance metric we mean the fraction of the probability of missing achieved by an algorithm to the probability of missing achieved by the optimal algorithm fed with perfect input information. Again, red cyclic points present the relative performance achieved by the heuristic swapping method and with the black-star line we denote the greedy algorithm, when the information of the popularity of request vectors was perfect. Black-cross points present the relative performance achieved by the heuristic swapping method and with the cyan-dotted line we denote the greedy algorithm, when the information of the popularity of requests vectors was imperfect, by a factor $\mu = 3$.

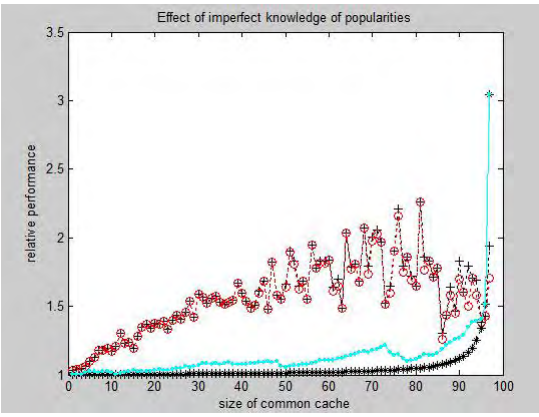

Figure 25

Figure 26 shows the mean value of relative performance values achieved by each of the heuristic algorithms over the variation of size of common cache, for increasing values of factor of distortion μ ($\mu = 1$ means that no distortion happened to popularity of request vectors). The blue bars represents the relative performance of greedy algorithm and the brown represents the relative performance of swapping algorithm.

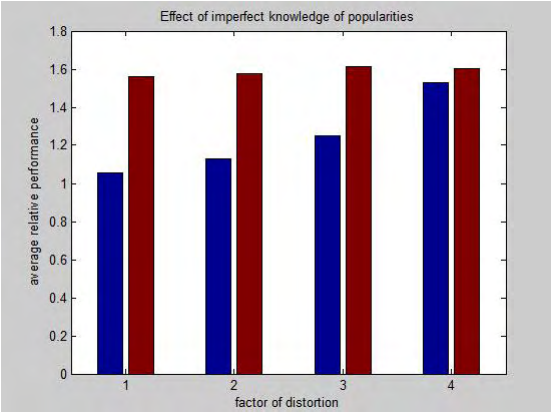

Figure 26

IX.e Joint Content placement and client assignment problem

We performed simulation on a topology of one parent cache of size 1 and five leaf caches allowing inter level cache cooperation, the number of object was constant and equal to $|O|$=100, the number of clients was 100, the rate of requests of all the stream is equal and normalized to 1 and popularity vectors are picked according to zipfian distribution where skew parameter is set $s = 0.8$ for all the clients.

Figure 27.a shows the probability of missing of the system for increasing values of sizes of leaf caches achieved by application of the clustering algorithm described in section VIII.b, that finds a client assignment to leaf caches, and then application of optimal Dynamic algorithm that finds the optimal content placement in caches given the above client assignment (blue line) compared to the probability of missing achieved if we keep the random client assignment and do not perform clustering and then apply Dynamic algorithm (red line). Figure 27.b differs from 28.a only in that the content placement algorithm that is used is Greedy instead of Dynamic, i.e. it shows the probability of missing of the system for increasing values of sizes of leaf caches achieved by application of the clustering algorithm, that finds a client assignment to leaf caches, and then application of Greedy algorithm that finds a suboptimal content placement in caches given the above client assignment (blue line) compared to the probability of missing achieved if we keep the random client assignment and do not perform clustering and then apply Greedy algorithm (red line).

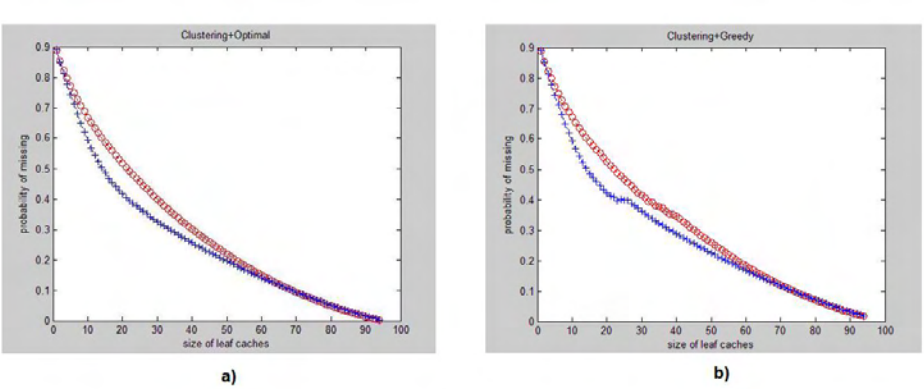

Figure 27

Figure 28 shows the average difference in probability of missing for all the sizes of leaf caches of the method of performing client assignment by clustering algorithm and content placement by Dynamic algorithm and the method that does client assignment at random and then performs Dynamic algorithm for content placement, for increasing values of zipfian parameter. We observe that as the zipfian parameter increases, the benefit of using clustering algorithm for performing client assignment instead of random algorithm increases.

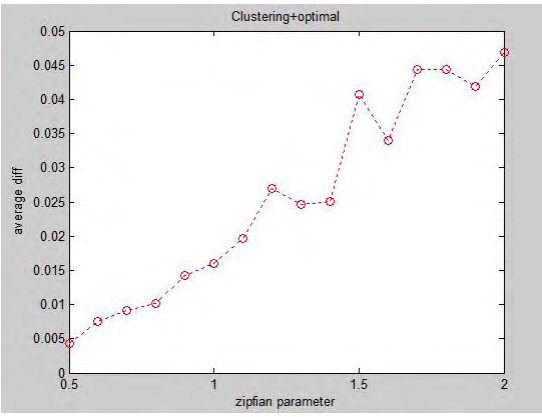

Figure 28

Figure 29 shows the number of iterations needed by the clustering algorithm to converge for increasing values of number of objects and number of clients respectively. We observe that the number of iterations does not get higher as the number of objects and/or streams scales. However the complexity of the clustering algorithm increases linearly to the number of objects and number of streams.

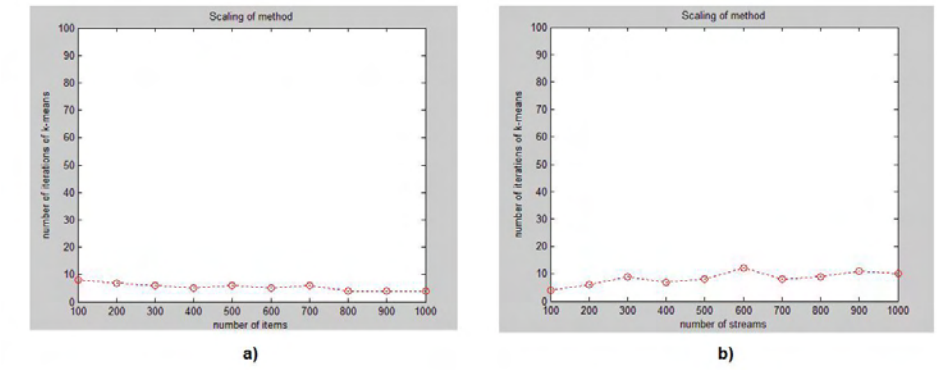

Figure 29

X. Conclusion and future work

Cooperative caching is an efficient way to exploit cache capacity and achieve content delivery to users at low delays and with low congestion of the network. In this study , we deal with the problems of content placement in caches and of assignment of clients in caches, assuming the partial replication model. We focused on hierarchical cache topologies and allowed inter-level cache cooperation for responding to requests generated by clients at the leaf nodes. We used as performance metrics the average rate at which origin server is accessed and the average cost emerged by the traversing of requests on the route to the origin server until they reach a cache that stores the requested content and showed that the first metric is a subcase of the second.

Content placement problem, assuming the partial replication model, is NP-complete problem in any topology, including the hierarchical one. However as we showed in this paper, the simplification that the size of each object is equal to 1, makes the problem polynomially solvable assuming that the cache topology is hierarchical and inter-level cache cooperation is permitted. The restriction of unit sized objects does not alleviate the applicability of the model, as objects of a real system could be split in smaller segments of data of the same popularity, so that all objects are of the same size and then perform the optimization procedure. We presented two optimal polynomial algorithms MWPMCP and RecDynamic. The first solves the problem only when restricting that the number of levels of the hierarchy is two, the number of leaf caches is two and the performance metric is the average rate at which origin server is accessed and the second that is applicable for any number of levels of the hierarchy and any number of leaf caches and uses the more general optimization metric of the average access cost of traversing of requests for content. Besides that, we established simulation comparative results between two well known heuristic suboptimal algorithms and the optimal presented algorithm in cases of perfect or imperfect information.

We also studied the problem of jointly performing content placement in caches and assignment of mobile clients in leaf caches in case of hierarchical topologies. We saw that this problem is solvable in case of non-cooperative model, by applying a subgradient method. We, further, investigated this problem in case that inter-level cache cooperation is allowed and proposed an algorithm that solves the two problems separately. Particularly we proposed a heuristic algorithm for the solution to the client assignment problem that groups the clients to clusters, depending on the similarity of the request patterns of them. The number of clusters is equal to the number of leaf caches and the input of two clients in the same cluster means that they are assigned to the same leaf cache. After client assignment is performed we apply RecDynamic Algorithm in order to perform content placement, given the requests on each leaf cache computed by the clustering procedure.

A topic for future work could be the treatment of cases that request rates per objects are not constant, but change dynamically over time, creating the need for online object replacement policies. Besides that, to the best of our knowledge, joint content placement and assignment of clients to caches in hierarchical cache topologies, where cooperation between caches is allowed (inter level or intra level) is still an open issue and it can be a topic for further study.

A. Appendix

Theorem 4: $C_0^*(n-1,O)$ is a subset of $C_0^*(n,O)$.

Proof

Recall that $C_0^*(n,O)$ is the subset of size n of object set O, that corresponds to the optimal content placement of the parent cache, when $|C_0|=n$, and the sizes of the leaf caches are constant $|C_1|,..,|C_N|$. We set $max_s(n,O)$ the n most popular objects in set O according to popularity vector p_s. We define as $H(P_0,O)$ the probability of hitting of the system that corresponds to the content placement in caches such that: $C_0=P_0$, $C_s=max_s(|C_s|,O\setminus P_0)$, i.e. the $|C_s|$ most popular objects according to vector p_s such that $P_0\cap C_s=\emptyset$, for s=1,..,N, when object set is O.

We will show that $C_0^*(n-1,O)$ is a subset of $C_0^*(n,O)$, by proving that every solution $C_0^*(n,O)$ that contains set $J_{n-1}\subset O$ of size n-1, instead of $C_0^*(n-1,O)$ gives lower probability of hitting, i.e. gives larger probability of missing.

Without lose of generality assume that $J_{n-1}=(C_0^*(n-1,O)\setminus i)\cup i'$ and $i=C_0^*(1,O\setminus C_0^*(n-2,O))$, i.e. set J_{n-1} and $C_0^*(n-1,O)$ to differ only in one element, where i was the $(n-1)^{th}$ object placed in common cache by the proposed algorithm and i' is the object that J_{n-1} contains instead of i. We define $\sigma=C_0^*(1,O\setminus {C_0}^*(n-1,O))$ and $\sigma'=C_0^*(1,O\setminus({C_0}^*(n-2,O)\cup i'))$. So it will be sufficient to show that always holds:

$H((i,\sigma),O\setminus C_0^*(n-2,O))\geq H((i',\sigma'),O\setminus C_0^*(n-2,O))$.

For simplicity assume that the number of streams of requests is two. We study the following subcases:

a) $i'\notin max_1(|C_1|,O\setminus C_0^*(n-1))\cup max_2(|C_2|,O\setminus C_0^*(n-1))$

It holds:

$H(i,O\setminus C_0^*(n-2,O))\geq H(i',O\setminus C_0^*(n-2,O))$ (2), because $i=C_0^*(1,O\setminus C_0^*(n-2,O))$.

We calculate:

$H((i,\sigma),O\setminus C_0^*(n-2,O))=H(i,O\setminus C_0^*(n-2,O))+\lambda_1 p_m^1+\lambda_2 p_m^2$, where

$$p_m^s=\begin{cases}p_\sigma^s & \text{if } \sigma\notin C_s(i,O\setminus C_0^*(n-2,O))\\ p_{m_s}^s & \text{else}\end{cases}$$

,where m_s is the most popular object in $O\setminus(C_0^*(n-1,O)\cup max_s(|C_s|,O\setminus C_0^*(n-1,O)))$ according to popularity vector p^s, for $s=1,2$.

it holds:

$H((i',\sigma'),O\setminus C_0^*(n-2,O))$ =

$H(i',O\setminus C_0^*(n-2,O))+\lambda_1 p_i^1+\lambda_2 p_i^2$ =

$H(i,O\setminus C_0^*(n-2,O))+\lambda_1 p_{i'}^1+\lambda_2 p_{i'}^2\leq$

$H(i,O\setminus C_0^*(n-2,O))+\lambda_1 p_{m_1}^1+\lambda_2 p_{m_2}^2=H((i,\sigma),O\setminus C_0^*(n-2,O))$.

,where the first equality comes from the fact that $\sigma'=i$ in this subcase and the inequality is from the definition of m_s. Figure 30.a. shows the selection of objects of the two popularity vectors in this subcase.

b) $i'\in max_1(|C_1|,O\setminus C_0^*(n-1))\cup max_2(|C_2|,O\setminus C_0^*(n-1))$.

Without lose of generality assume $i' \in max_1(|C_1|, O \backslash C_0^*(n-1))$ and $i' \notin max_2(|C_2|, O \backslash C_0^*(n-1))$, as shown in figure 30.b.
We study the following subcases of subcase b).
b.i)if $\sigma' \notin max_1(|C_1|, O \setminus C_0^*(n-1)) \cup max_2(|C_2|, O \setminus C_0^*(n-1))$

$H((i', \sigma'), O \setminus C_0^*(n-2, O))$ =
$\lambda_1 p_{i'}^1 + \lambda_2 p_{i'}^2 + H(\sigma', O \setminus (C_0^*(n-2, O) \cup i'))$=
$\lambda_1 p_{i'}^1 + \lambda_2 p_{i'}^2 + \lambda_1 p_{\sigma'}^1 + \lambda_2 p_{\sigma'}^2 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus (C_0^*(n-2) \cup i'))} p_j^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2$=

$\lambda_2 p_{i'}^2 + \lambda_1 p_{\sigma'}^1 + \lambda_2 p_{\sigma'}^2 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p_j^1 + \lambda_1 p_{m_{i'}}^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2 \leq$

$\lambda_1 p_{\sigma'}^1 + \lambda_2 p_{\sigma'}^2 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p_j^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2 +$
$\lambda_1 p_m^1 + \lambda_2 p_m^2 \leq$

$H(i, O) + \lambda_1 p_{\sigma'}^1 + \lambda_2 p_{\sigma'}^2 \leq$
$H((i, \sigma), O \setminus C_0^*(n-2, O))$

,where $m_{i'} = max_1(1, O \setminus max_1(|C_1|, O \setminus C_0^*(n-1))$
b.ii)if $\sigma' \in max_1(|C_1|, O \setminus C_0^*(n-1)) \cup max_2(|C_2|, O \setminus C_0^*(n-1))$
Without lose of generality assume $\sigma' \notin max_1(|C_1|, O \backslash C_0^*(n-1))$ and $\sigma' \in max_2(|C_2|, O \backslash C_0^*(n-1))$.

It holds:
$H((i', \sigma'), O \setminus C_0^*(n-2, O))$ =
$\lambda_1 p_{i'}^1 + \lambda_2 p_{i'}^2 + \lambda_1 p_{\sigma'}^1 + \lambda_2 p_{\sigma'}^2 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus (C_0^*(n-2)) \cup i' \cup \sigma')} p_j^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i' \cup \sigma'))} p_j^2 =$

$\lambda_2 p_{i'}^2 + \lambda_1 p_{\sigma'}^1 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p_j^1 + \lambda_1 p_{m_{i'}}^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2 +$
$\lambda_2 p_{m_{\sigma'}}^2$ (3)

,where $m_{i'} = max_1(1, O \setminus max_1(|C_1|, O \setminus C_0^*(n-1))$
and $m_{\sigma'} = max_2(1, O \setminus max_1(|C_1|, O \setminus (C_0^*(n-2) \cup i'))$
We study the following four subcases due to the values of p_i^s and p_m^s, $s = 1, 2$.
1)if $p_i^1 > p_{m_1}^1$ and $p_i^2 > p_{m_2}^2$
then $m_{i'} = i$, $\sigma' = m_1$, $m_{\sigma'} = i$.
$H((i', \sigma'), O \setminus C_0^*(n-2, O))$ =
$\lambda_2 p_{i'}^2 + \lambda_1 p_{m_1}^1 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p_j^1 + \lambda_1 p_i^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2 +$
$\lambda_2 p_i^2 \leq \lambda_2 p_{m_2}^2 + \lambda_1 p_{m_1}^1 + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p_j^1 + \lambda_1 p_i^1 + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p_j^2 +$
$\lambda_2 p_i^2 =$
$H((i, \sigma), O \setminus C_0^*(n-2, O))$.

2)if $p_i^1 > p_{m_1}^1$ and $p_i^2 \leq p_{m_2}^2$
then $m_{i'} = i$, $\sigma' = m_1$, $m_{\sigma'} = m_2$.

$H((i',\sigma'), O \setminus C_0^*(n-2,O))$ =

$\lambda_2 p^2_{i'} + \lambda_1 p^1_{m_1} + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_i + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p^2_j + \lambda_2 p^2_{m_2} \leq$
$\lambda_2 p^2_{i} + \lambda_1 p^1_{m_1} + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_i + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p^2_j + \lambda_2 p^2_{m_2} =$
$H((i,\sigma), O \setminus C_0^*(n-2,O))$.

3)if $p^1_i \leq p^1_m$ and $p^2_i > p^2_m$
then $m_{i'} = m_1$, $\sigma' = sm_1$, $m_{\sigma'} = i$.

$H((i',\sigma'), O \setminus C_0^*(n-2,O))$ =

$\lambda_2 p^2_{i'} + \lambda_1 p^1_{sm_1} + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_{m_1} + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (^*_0(n-2) \cup i'))} p^2_j + \lambda_2 p^2_i \leq$

$\lambda_2 p^2_{i} + \lambda_1 p^1_i + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_{m_1} + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p^2_j + \lambda_2 p^2_{m_2} =$

$H((i,\sigma), O \setminus C_0^*(n-2,O))$.

That can be proved by considering that $i = C_0^*(n-2,O)$, which yields that $\lambda_1 p^1_i + \lambda_2 p^2_i \geq \lambda_1 p^1_{m_1} + \lambda_2 p^2_{i'}$, which yields that $\lambda_1 p^1_i + \lambda_2 p^2_{m_2} \geq \lambda_1 p^1_{m_1} + \lambda_2 p^2_{i'} - \lambda_2 p^2_i + \lambda_2 p^2_{m_2} \geq \lambda_2 p^2_{i'} + \lambda_1 p^1_{sm_1}$
4)if $p^1_i \leq p^1_m$ and $p^2_i \leq p^2_m$
then $m_{i'} = m_1$, $\sigma' = sm_1$, $m_{\sigma'} = m_2$.

$H((i',\sigma'), O \setminus C_0^*(n-2,O))$ =

$\lambda_2 p^2_{i'} + \lambda_1 p^1_{sm_1} + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_{m_1} + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p^2_j + \lambda_2 p^2_{m_2} \leq$
$\lambda_2 p^2_{i} + \lambda_1 p^1_i + \lambda_1 \sum_{j \in max_1(|C_1|, O \setminus C_0^*(n-2))} p^1_j + \lambda_1 p^1_{m_1} + \lambda_2 \sum_{j \in max_2(|C_2|, O \setminus (C_0^*(n-2) \cup i'))} p^2_j + \lambda_2 p^2_{m_2} =$

$H((i,\sigma), O \setminus C_0^*(n-2,O))$.

That can be proved by considering that $i = C_0^*(n-2,O)$, which yields that $\lambda_1 p^1_i + \lambda_2 p^2_i \geq \lambda_1 p^1_{m_1} + \lambda_2 p^2_{i'} \geq \lambda_1 p^1_{sm_1} + \lambda_2 p^2_{i'}$.

,where $sm_1 = max_1(1, O \setminus max_1(1, O \setminus (C_0^*(n-1)) \cup m_1)$

So, we proved that for every choice of objects i,i',σ and σ' it holds that: $H((i,\sigma), O \setminus C_0^*(n-2,O)) \geq H((i',\sigma'), O \setminus C_0^*(n-2,O))$

So, swapping an object $i \in C_0^*(n-1, O)$ with an object $i' \notin C_0^*(n-1, O)$, can only decrease probability of hitting of the system. The above arguments can be applied for any number of streams $N \geq 2$. Proof of theorem completed.

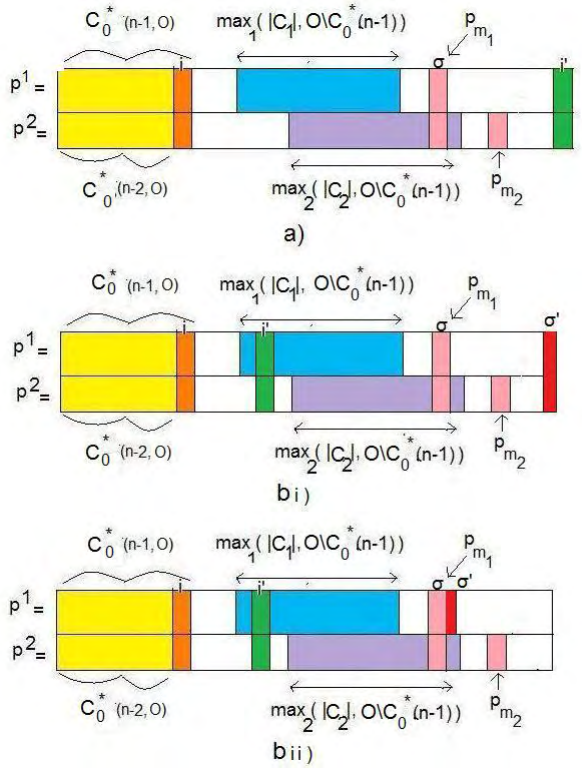

Figure 30

REFERENCES

[1] B. Li, M. Golin, G. Italiano, X. Deng, On the Optimal Placement of Web Proxies in the Internet. Infocom 1999, New York, US.

[2] L. Qiu, et al., On the Placement of Web Server Replicas. Infocom 2001, Alaska, US.

[3] P.Wolfe, and H. P. Crowder. Validation of Subgradient Optimization. Math Programming. Vol. 6, 1974, pp. 62 - 88.

[4] J. Kangasharju, J. Roberts, Keith Ross: Object Replication Strategies in Content Distributions Networks. Computer Communications, 2002.

[5] J. Sun, S. Gao, W. Yang, Z. Jiang: Heuristic Replica Placement Algorithms in Content Distribution Networks, Journal of networks, 2011.

[6] S. Khan, I. Ahmad. Heuristic-based replication schemas for fast information retrieval over the internet (Proceeding of the 17th international conference on parallel and distributed computing systems, 2004)

[7] T. Loukopoulos, I. Ahmad. Static and adaptive distributed data replication using genetic algorithms (J. parallel Distrib. Comput., 2004)

[8] M. Korupolu, G Plaxton, R. Rajaraman (Proc. SODA 99)

[9] N. Laoutaris, O. Telelis, V. Zissimopoulos, I. Stavrakakis: Distributed Selfish Replication (IEEE transactions on parallel and distributed systems, 2006)

[10] S. Borst. V. Gupta A. Walid: Distributed Caching Algorithms for Content Distribution Networks Infocom 2010

[11] T. Bektas, J. Cordeau, E. Erkut, G. Laporte: Exact algorithms for the joint object placement and request routing problem in CDNs (Computer and Operation research, 2008)

[12] S. Chen, T. Gu, X. Tao, J. Lu: Application based distance measurement for context retrieval in ubiquitous computing(2007)

[13] A. Leff, J.L. Wolf, P.S. Yu (1993). Replication algorithms in a remote caching architecture. IEEE Trans. Parallel Distr. Syst. 4 (11), 1185 1204.

[14] Laoutaris N, Zissimopoulos V, Stavrakakis I. Joint object placement and node dimensioning for Internet content distribution. Information Processing Letters 2004;89:2739.

[15] Laoutaris N, Zissimopoulos V, Stavrakakis I. On the optimization of storage capacity allocation for content distribution. Computer Networks 2005;47:40928.

[16] Almeida JM, Eager DL,Vernon MK,Wright SJ. Minimizing delivery cost in scalable streaming content distribution systems. IEEE Transactions on Multimedia 2004;6:35665.

[17] Nguyen TV, Safaei F, Boustead P, Chou CT. Provisioning overlay distribution networks. Computer Networks 2005;49:10318.

[18] Bektas T, Oguz O, Ouveysi I. Designing cost-effective content distribution networks. Computers and Operations Research 2007;34:243649.

[19] J. Kangasharju, J. Roberts, K. Ross, Object Replication Strategies in Content Distribution Networks (2002)

[20] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello. Push-to-peer video-on-demand system: Design and evaluation. IEEE Journal on Selected Areas in Communications, 25(9):17061716, 2007.

[21] Bo (Rambo) Tan, Laurent Massoulie. Optimal Content Placement for Peer-to-Peer Video-on-Demand Systems.

[22] Glover F. Improved linear integer programming formulations of nonlinear integer programs. Management Science 1975;22:45560.