A Distributed Algorithm for Sharing Web Cache Disk Capacity

George Alyfantis, Stathes Hadjiefthymiades, Lazaros Merakos, and Panagiotis Kostopoulos

Communication Networks Laboratory

Department of Informatics and Telecommunications, University of Athens

Athens 15784, Greece

{alyf, shadj, merakos, p.kostopoylos}@di.uoa.gr

Abstract

A decentralized game theoretic framework applied to Web caching is discussed. The interaction of multiple clients with a caching server is modeled as a non-cooperative game, in which clients are viewed as players, and the caching server disk space as a resource for which players are competing. However, some clients may continuously request new objects, thus, occupying a considerable portion of the cache disk, enjoying high hit rates. Such an aggressive behavior may have significant impact to the overall cache performance, as few clients may monopolize the total disk space, and the remaining clients may suffer the eviction of their "important" resources from the cache, thus, experiencing numerous cache misses. Moreover, it is observed that the majority of hits in cache systems are due to shared objects, i.e., clients tend to refer to a "pool" of common resources. The objective of the proposed framework is to discourage monopolizing the cache disk space by a minority of clients, while rewarding clients that contribute to the overall hit rate. The efficiency of the proposed scheme is evaluated through simulations.

1. Introduction

The use of web caches for reducing network traffic and download latency has rendered them an important component of the Internet infrastructure. Algorithms devised for web caches need to exploit the specific characteristics present in the request sequence for efficient performance. For example, it is observed from real proxy traces that

some clients, exhibiting "aggressive" behavior, tend to monopolize the cache disk space, enjoying high hit rates [10]. However, at the same time, the other clients are confined to a restricted amount of disk space, and suffer the eviction of "important" objects, thus, experiencing numerous cache misses. Conventional replacement algorithms [9] apply global policies, i.e., treat all clients as a whole, without taking into account the behavior of individual clients, thus, proving incapable of providing the caching service in a fair manner.

As reported in [1], cache hits are mostly due to shared objects ("sharing hits"); the rest are termed "locality hits". That is to say, it is observed that clients tend to request objects that belong to a popular "pool" of objects, while the rest of the objects are rarely requested more than once. In this paper, we discuss a distributed, game theoretic, algorithmic framework capable of providing the caching service to the clients in a fair way, while also taking into account the contribution of each client to the overall performance of the caching system. Specifically, with regards to the latter, the proposed system rewards clients that have fetched popular objects into the cache, allowing them to use more disk space. In this paper we assume that the caching service is provided on a commercial basis (e.g., by the ISP) and clients are charged for exploiting the resources (disk space) of such a commercial entity [12].

Clients are considered selfish, i.e., they do not collaborate with one another. There is not a single coordinating entity regulating the access to the cache. The caching server communicates global information on the state of the cache to the clients, e.g., free space currently left, and client specific information, e.g., the space that the client occupies into the cache. This information is delivered to the client embedded in HTTP responses. The next time the client issues a request, he takes into account the information acquired from the previous HTTP response, in order to determine whether it is worth requesting the caching of the requested object, or the induced cost is higher than the corresponding benefit. In the latter case, the client sets the HTTP request header "cache-control: no-store" [9] denoting that, if the object is not already into the cache, the object that will be fetched from the origin server shall not be cached.

The rest of the paper is structured as follows. We formulate the problem as a non-cooperative game in Section 2. In Section 3, we study the existence and uniqueness of the Nash equilibrium. In Section 4, we provide an iterative algorithm for decentralized convergence to the Nash equilibrium. In Section 5, we propose a variation of the theoretical iterative algorithm appropriately adapted to the caching game. Section 6 provides simulation results. In Sec-

tion 7, we discuss prior related work. Section 8 concludes the paper with a summary of our key findings as well as directions for future work in this area.

2. Model Formulation

Let $I = \{1, ..., N\}$ be the set of clients who share the resources of a caching server with disk capacity C. Client i controls the cache capacity y_i that he occupies so that he optimizes a certain performance measure. We assume that y_i is chosen from $S_i = [0,C]$. Let $\boldsymbol{y} = (y_1, \ldots, y_N)$ be a typical strategy profile (or cache disk space allocation) vector in the strategy space $S = S_1 \times \ldots \times S_N$ and $F = \sum_{i=1}^{N} y_i$ the total reserved cache capacity.

We assume that each client has a utility function. The utility function consists of *benefit* and *cost* parts, and quantifies the level of user satisfaction for using the caching server. The cost function is influenced by the characteristics of the caching system, e.g., the adopted pricing scheme. Each client is capable of computing his cost by some measurement or feedback information from the system.

The utility function u_i: $S \to \Re$ of client i depends on the strategy profile $\boldsymbol{y}$ only through y_i and F. We denote this by $u_i(y_i, F)$. Client i may maximize $u_i(y_i, F)$ by controlling y_i, given the total capacity C and the "aggregate" strategy F of all clients. We assume that $u_i(y_i, F)$ must have the following properties:

Al. $u_i(y_i, F)$ is continuously differentiable with respect to y_i.

A2. $\partial u_i(y_i, F)/\partial y_i$ is strictly decreasing with respect to y_i.

A3. $\partial u_i(y_i, F)/\partial F$ is non-increasing with respect to F.

The first two assumptions imply the strict concavity of u_i with respect to y_i. Let $C_i = C - \sum_{j \neq i} y_j$ be the available cache capacity seen by client i. Note that $F = y_i + C - C_i$. Then, the response of client i for a given available capacity is defined as

$$r_i(C_i) = \arg\max_{g \in S_i} u_i\left(g, g + C - C_i\right). \qquad (1)$$

In other words, $r_i(C_i)$ is the optimal amount of cache space client i should occupy, which maximizes u_i, if the available capacity for i is C_i. We also make the following natural assumption on the response function of the client.

A4. Given any C_i such that $0 < C_i \leq C$, $r_i(C_i) < C_i$

The above assumption holds true for many situations, where the cost function has a form of penalty (e.g., price) that prevents the saturation of critical resources. An example utility function that is inline with the four assumptions is the generalized power [2]:

$$u_i(y_i, F) = (y_i)^{\beta_i}(C - F) \qquad (2)$$

which is interpreted as benefit $(y_i)^{\beta_i}$ divided by cost $(C\text{-}F)^{-1}$. This type of utility function satisfies all four assumptions A1-A4, if $0 < \beta_i \leq 1$.

In this paper, we propose a similar utility function that also takes into account the impact that a client has on the overall performance of the caching system. Specifically, client i is associated with *popularity index* a_i. This index is calculated dynamically by the caching server and denotes the "popularity" of the objects that client i has fetched into the cache. The popularity index for client i, a_i, assumes values in the interval [0,1).

We assume that each cached object carries some meta-information to facilitate system operation. Specifically, for each cached object $j \in \{1, \ldots, K\}$, the caching server needs to know the client i_j that brought the object into the cache, the size s_j of the object in bytes, and the number h_j of times that the object was hit by clients other than client i_j. Then, the popularity factor a_i for client i is calculated as

$$a_i = \begin{cases} \sum_{j \in I_i} h_j \cdot s_j \Big/ \sum_{j=1}^{K} h_j \cdot s_j, & \sum_{j=1}^{K} h_j \cdot s_j \neq 0 \\ 0, & otherwise \end{cases} \qquad (3)$$

where I_i is the set of objects that client i has brought into the cache.

We now define the utility function of client i as

$$u_i(y_i, F) = (y_i)^{\beta_i}(C - F)^{1-a_i}. \qquad (4)$$

The proposed utility function is of similar form to (2), apart from the cost part. Specifically, the cost of client i is discounted by the factor $1\text{-}a_i$. In other words, "popular" clients are charged less than clients that are not popular, which gives them more freedom to use the disk space of the caching server.

Proposition 2.1: The utility function $u_i(\cdot)$, for each $i \in I$, satisfies assumptions A1-A4.

Proof:

Assumption A1

It can be easily shown that the derivative of $u_i(\cdot)$ with respect to y_i, for each $i \in I$, is a continuous function of y_i on S_i.

Assumption A2

The second derivative of $u_i(\cdot)$ with respect to y_i, for $i \in I$, is negative, which means that A2 also holds.

Assumption A3

The second derivative of $u_i(\cdot)$ with respect to F is non-positive, thus, A3 holds.

Assumption A4

Given a C_i, the optimality condition $\nabla u_i = 0$ yields

$$\arg\max_{y_i \in S_i} u_i(y_i, F) = \frac{\beta_i}{1 + \beta_i - a_i} C_i < C_i$$

which means that A4 also holds, and the result of the proposition follows. ■

Clients are selfish, implying that each client is interested only in maximizing his utility function. We assume that u_i is private information for client i, while other information (e.g., F, C) is public, provided by the caching server. Then, given F, each client computes his best strategy by solving the following optimization problem (P).

$$\text{(P)} \quad \begin{cases} \max\limits_{y_i} \; u_i(y_i, F) \\ s.t. \\ y_i \in S_i, \; i \in I \end{cases}$$

The fact that u_i depends on $\boldsymbol{y}$ only through y_i and $\sum_{j \neq i} y_j$, drives client i to consider all other clients as a single adversary, and, thus, greatly simplifies the analysis. The vector $\boldsymbol{y}^* = (y_1^*, \ldots, y_N^*) \in S$ is said to constitute a Nash Equilibrium (NE) [3] iff

$$y_i^* \in \arg\max_{g \in S_i} u_i\left(g, g + \sum_{j \neq i} y_j^*\right), \quad \forall i \in I \,. \quad (5)$$

In other words, at the NE, given the disk space allocated to the other clients, no client can improve his utility by making individual changes to his disk space allocation. In the following section, we study the existence and uniqueness of a NE.

3. Existence and Uniqueness of the Nash Equilibrium Point

Due to the properties of the utility function, discussed in the previous section, the existence of the NE can be guaranteed from Theorem 1 in [4]. For any NE, the set of fixed-point equations, deriving from the optimality condition $\nabla u_i = 0$, can be written as a system of equations, as follows:

$$\begin{bmatrix} \beta_1 + 1 - a_1 & \beta_1 & \dots \\ \beta_2 & \beta_2 + 1 - a_2 & \dots \\ \dots & \dots & \dots \\ \beta_N & \dots & \dots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix} = \begin{bmatrix} \beta_1 C \\ \beta_2 C \\ \dots \\ \beta_N C \end{bmatrix} \qquad (6)$$

The uniqueness of the NE can be proven if we consider the properties of the particular system.

Proposition 3.1: There exists one and only NE $\boldsymbol{y}^* = (y_1^*, \dots, y_N^*)^T$ for the game, which is feasible, i.e., $\sum_{i=1}^{N} y_i^* < C$, with $y_i^* = C\left(\frac{\beta_i}{1-a_i}\right)\left[1 + \sum_{j=1}^{N}\left(\frac{\beta_j}{1-a_j}\right)\right]^{-1}$.

Proof: The uniqueness of the NE stems from the fact that the system of fixed-point equations is linear and the matrix of coefficients is *full rank*. The solution to the system $\boldsymbol{y}^* = (y^*_1, y^*_2, \dots, y^*_N)^T$ is given as

$$y_i^* = C\left(\frac{\beta_i}{1-a_i}\right)\left[1 + \sum_{j=1}^{N}\left(\frac{\beta_j}{1-a_j}\right)\right]^{-1}, \text{ for each i} \in I.$$

Then, by adding the elements of the solution vector, we obtain that

$$\sum_{i=1}^{N} y_i^* = C\left(\sum_{i=1}^{N}\frac{\beta_i}{1-a_i}\right)\left[1 + \sum_{j=1}^{N}\left(\frac{\beta_j}{1-a_j}\right)\right]^{-1} < C$$

Hence, the solution to the system is feasible and the result of the proposition follows. ■

Next, we consider fairness issues regarding the discussed NE. In this paper, we assume that fairness implies that a client that is in greater "need" for cache disk space will indeed get more disk space at the NE. We define fairness of the NE in terms of the client's response function as follows (see also [5]).

Definition 3.1: Client i is in "greater need" for disk space than client j, if $r_i(C') \geq r_j(C')$ for all available capacity $C' > 0$. The NE $\boldsymbol{y}^* = (y_1^*, \ldots, y_N^*)^T$ is said to be fair if, for every i and j, such that i is in greater need than j, $y_i^* \geq y_j^*$.

Lemma 3.1: For any i, both $r_i(C')$ and $C' - r_i(C')$ are monotone increasing for $C' > 0$.

Proof: As shown in the proof of Proposition 2.1, the response function of client i is given as

$$r_i(C') = \frac{\beta_i}{1 + \beta_i - a_i} C' .$$

For $0 \leq a_i < 1$, $0 < \beta_i \leq 1$, it is easy to see that $\beta_i/(1 + \beta_i - a_i) > 0$, thus, the response function of client i is monotone increasing. We now see that

$C' - r_i(C') = \frac{1 - a_i}{1 + \beta_i - a_i} C'$, thus, the function $C' - r_i(C')$ is also monotone increasing. ■

Theorem 3.1: The NE of the decentralized cache disk space allocation that is given by (P) is fair.

Proof: Consider two clients i and j such that $r_i(C') \geq r_j(C')$ for all available disk space $C' > 0$. By the definition of the NE and $r_i(\cdot)$, we have that

$$y_i^* = \arg\max_{g \in S_i} u_i\left(g, g + C - C_i^*\right) = r_i\left(C_i^*\right) \geq r_j\left(C_i^*\right).$$

Since $C_i^* = C - \sum_{j \neq i} y_j^* = C - F^* + y_i^*$, i.e.,

$y_i^* = C_i^* - C + F^*$, the above inequality can be rewritten as $C_i^* - r_j\left(C_i^*\right) \geq C - F^*$. Note that $C - F^* > 0$ by Proposition 3.1. From $y_j^* = r_j\left(C_j^*\right)$, we have $C_j^* - r_j\left(C_j^*\right) = C - F^*$

By the monotone increasing property of function $C' - r_i(C')$ in Lemma 3.1, $C_i^* \geq C_j^*$, and, thus, $y_i^* = F^* - C + C_i^* \geq F^* - C + C_j^* = y_j^*$. ■

4. Convergence to the Nash Equilibrium Point

In this section, we describe an iterative Gauss-Seidel scheme for convergence to the NE. This scheme assumes that clients operate iteratively in an asynchronous fashion. Following HTTP interactions, clients update their reserved disk space, in order to maximize their utility functions with respect to the current state of the system. This is compatible with the considered WWW caching scenario, where clients can only update their information on the status of the system, after receiving an HTTP response.

At each step of the Gauss-Seidel scheme (i.e., each time a client performs a HTTP interaction), the client is informed on the overall state of the system, as well as on his own status, and re-computes his optimal reserved disk space. Only the client that interacts with the caching server updates his strategy. Such algorithms are called *nonlinear* or *coordinate descent* [6] . The main idea behind this class of algorithms is that we fix all the components of $\mathbf{y}$, except for the ith component, and then maximize $u_i(\mathbf{y})$ with respect to y_i. This procedure is repeated, thus, leading to an iterative algorithm.

In the *nonlinear Gauss-Seidel* algorithm, the maximizations are carried out successively for each component. Formally, the algorithm can be described as follows:

$$y_i(t+1) = \arg\max_{y_i} u_i\left(y_1(t+1),...,y_{i-1}(t+1), y_i, y_{i+1}(t),...,y_N(t)\right) \qquad (7)$$

In (7), it is implied that the order by which clients update their strategies is predefined. However, this is not obligatory. Order may be chosen randomly and changed dynamically [6]. Below, we show that, under the Gauss-Seidel scheme, the vector of disk space capacities reserved by all clients, at step t, $\mathbf{y}(t) = (y_1(t), \ldots, y_N(t))^T$, converges to the unique NE $\mathbf{y}^* = (y_1^*, \ldots, y_N^*)^T$, as $t \to \infty$. By (5), it is easy to show that client i, at step t+1, given the strategies of the other clients should respond as follows:

$$y_i(t+1) = \frac{\beta_i}{1+\beta_i - a_i}\left(C - \sum_{j \neq i} y_j(t)\right) \qquad (8)$$

Let $F(t) = \sum_{j=1}^{N} y_j(t)$ be the total disk space reserved by all clients after step t, and define the following metric:

$$S(\mathbf{y}(t)) = \sum_{j=1}^{N} \left| y_j(t) - y_j^* \right| + \left| F(t) - F^* \right|$$

It will be shown that $S(\boldsymbol{y}(t))$ converges to 0 (i.e., individual strategies converge to the NE). We begin with the following lemma.

Lemma 4.1: Assume that client i performs the $(t+1)$th step, then

$$F(t+1) \geq F^* \Rightarrow y_i(t+1) \leq y_i^*$$

and, conversely

$$F(t+1) \leq F^* \Rightarrow y_i(t+1) \geq y_i^*$$

where

$$F^* = \sum_{j=1}^{N} y_j^* = C\left(\sum_{j=1}^{N} \frac{\beta_j}{1-a_j}\right)\left[1+\sum_{j=1}^{N} \frac{\beta_j}{1-a_j}\right]^{-1}.$$

Furthermore, the above also holds true when all inequalities are strict.

Proof: By (8), the total disk capacity reserved by all clients after step $(t+1)$ will be

$$F(t+1) = \sum_{j \neq i} y_j(t) + \frac{\beta_i}{1+\beta_i - a_i}\left(C - \sum_{j \neq i} y_j(t)\right)$$

or

$$F(t+1) = \frac{\beta_i}{1+\beta_i - a_i} C + \frac{1-a_i}{1+\beta_i - a_i} \sum_{j \neq i} y_j(t)$$

If we now assume that $F(t+1) \geq F^*$, it is implied that

$$\sum_{j \neq i} y_j(t) \geq$$

$$\geq \frac{1+\beta_i - a_i}{1-a_i} C\left(\left(\sum_{j=1}^{N} \frac{\beta_j}{1-a_j}\right)\left[1+\sum_{j=1}^{N} \frac{\beta_j}{1-a_j}\right]^{-1} - \frac{\beta_i}{1+\beta_i - a_i}\right)$$

so it follows that

$$y_i(t+1) = \frac{\beta_i}{1+\beta_i - a_i}\left(C - \sum_{j \neq i} y_j(t)\right)$$

$$\leq \frac{\beta_i}{1-a_i} C\left[1+\sum_{j=1}^{N} \frac{\beta_j}{1-a_j}\right]^{-1} = y_i^*$$

Similarly, it can be shown that $y_i(t+1) \geq y_i^*$, if $F(t+1) \leq F^*$. For the case of strict inequalities the proof remains the same. ■

The next lemma states that the sequence $S(\mathbf{y}(t))$, $t \in \aleph$, is non increasing [7].

Lemma 4.2: Under the Gauss-Seidel scheme

$$S(\mathbf{y}(t+1)) \leq S(\mathbf{y}(t)), \text{ for all } t \in \aleph.$$

The following lemma states that the sequence $S(\mathbf{y}(t))$ strictly decreases after a finite number of steps [7].

Lemma 4.3: For all t there exists a (finite) $k > t$, such that, if $(y_1(t), \ldots, y_M(t)) \neq (y_1^*, \ldots, y_N^*)$, then $S(\mathbf{y}(k)) < S(\mathbf{y}(t))$.

The convergence of the Gauss-Seidel to the unique NE is stated in the following theorem [7].

Theorem 4.1: Under the Gauss-Seidel scheme, the vector of cache disk space reserved by all clients at step t, $\mathbf{y}(t) = (y_1(t), \ldots, y_N(t))^T$, converges to the unique NE $(y_1^*, \ldots, y_N^*)$ as $t \to \infty$, i.e., $\lim_{t \to \infty} \mathbf{y}(t) = \mathbf{y}^*$.

5. Distributed Algorithm Implementation

In the previous section, we showed that the Gauss-Seidel nonlinear iterative scheme converges to the NE. However, such scheme cannot be applied unmodified in the caching game, due to the following reasons: 1) a client by performing an individual HTTP interaction can only demand (or not) the caching of the corresponding requested object (i.e., the client can increase his reserved disk space by one object at a time), and 2) clients can only increase their disk space, as if they exceed their equilibrium reservation, they are not capable of "shrinking" their area in the disk, unless the disk space is full and the replacement algorithm (e.g., Least Recently Used – LRU) is invoked. For this reason, we provide a variation of the scheme introduced in the previous section, taking into account the technical constraints.

5.1 Algorithm Variation

As an indicative example, consider that client i, at step t, uses the response function in (8), with the information provided at the previous HTTP response (free disk space $C - F$, popularity index a_i, and reserved disk space y_i), in order to update his reserved capacity $y_i(t)$. If the response function indicates that the client should respond with a

reserved capacity greater than his current reserved capacity, the client could only increase his disk space by the size of the requested object, i.e., he cannot "fully" reply to the strategies of the other clients. Reversely, if the response function instructs the client to reduce his occupied disk space, the client could only order the caching server not to cache the requested object, i.e., keeping his reserved disk space constant.

Based on the above, we propose a variation of the iterative nonlinear Gauss-Seidel scheme. Let the initial vector of reserved capacities for all clients be $\boldsymbol{y}(0) = (0, \ldots, 0)^T$. Then, client i, at step t+1, given the delivered information after the last HTTP response, computes his best response $r_i(t+1) = \beta_i(1 + \beta_i - a_i)^{-1}(C - F(t) + y_i(t))$ with respect to the state of the system. Then, if $r_i(t+1) > y_i(t)$, the client instructs the caching server to store the requested object (if not already cached). Otherwise, the client instructs the caching server simply to fetch the object, but not to cache it. The caching server returns the requested object and also updates the client on the current system state.

5.2 Implementation Issues

In this section, we discuss how the proposed asynchronous distributed algorithm can be implemented. We first describe the requirements for a caching server to support the proposed functionality. For each cached object the caching server maintains the following meta-information: 1) the ID of the client that triggered the caching of the resource, 2) the number of times the object was hit by other clients, and 3) the size of the object. We can represent a cached object through the record shown in Fig. 1.

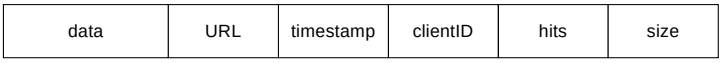

Figure 1. Structure of a cached object

Information on each client of the caching server needs also to be maintained. Specifically, the caching server has to be up to date with the space that every client occupies, and his popularity factor (i.e., the numerator of the popularity index in (3)). Hence, the caching server holds a record for each client as shown in Fig. 2.

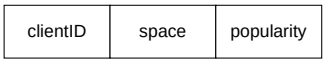

Figure 2. Structure of a client record

We also assume that the caching server is capable of performing the following three operations:

1) lookup for a specific object in the cache using its URL,

2) lookup for the least recently used object into the cache, and,

3) lookup for a specific client record by his ID.

Before presenting the algorithms run by the caching server and the clients, we define the structure of the request and response messages. Fig. 3 and Fig. 4 show the structure of the request and the response message, respectively.

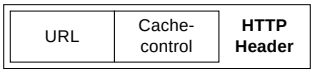

Figure 3. Structure of request message

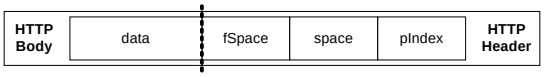

Figure 4. Structure of response message

In Fig. 4, *fSpace* denotes the cache free disk space, *space* the disk space occupied by the client that issued the corresponding request, and *pIndex* is the popularity index of the specific client. All these header fields can be integrated in the response message through the HTTP extension framework [11].

Subsequently, we present the algorithms executed by the caching server. The actions taken by the caching server, when the replacement algorithm is invoked, are presented in Listing 1.

```
Function invokeLRU()

// Specify the target free space
targetFreeSpace = LRUFactor*C

do
 // find the least recently used object
 LRU_obj = LRUObjectLookup()
 s = LRU_obj->size
 client_ID = LRU_obj->clientID
 hits_num = LRU_obj->hits
 client = clientLookup(client_ID)
 // update the client's popularity
 client->popularity -= hits_num*s
 // update the client's occupied disk space
 client->space -= s

 cacheFreeSpace += s
 remove(LRU_obj)
while (cacheFreeSpace < targetFreeSpace)

endFunction
```

Listing 1. Replacement algorithm

Upon receipt of a request from a client, the caching server performs the actions described by the pseudo code in Listing 2.

```
Event receiveRequest (req)

requestorID = getClientID(req)
url = req->URL
cache_instruction = req->cache_control
requested_obj = cacheLookup(url)
//object found in cache
if (requested_obj != NULL) then
 requested_obj->hits++
 client_ID = requested_obj->clientID
 client = clientLookup(client_ID)
 s = requested_obj->size
 client->popularity += s
else //object not found in cache
 requested_obj = fetchObject(URL)
 // client instructs for caching
 if (cache_instruction == TRUE) then
  requested_obj->clientID = requestorID
  requested_obj->hits = 0
  s = requested_obj->size
  // No adequate free disk space
  if (cacheFreeSpace – s < 0) then
   invokeLRU()
  endif
   cacheObject(requested_obj)
   // store the object into the cache
   client = clientLookup(requestorID)
   client->space += s
 endif
endif

reply->data= requested_obj->data
reply->fSpace = cacheFreeSpace
client = clientLookup(requestorID)
reply->space = client->space
reply->pIndex = client-
>popularity/requestOverlapping
sendReply(requestorID,reply)

endEvent
```

Listing 2. Request handling at the caching server

6. Simulation Results

We have tried to assess the impact of the proposed game theoretic mechanism on a caching setting involving the interaction of a number of clients with a single cache server. We have also simulated the non-game theoretic scenario. Our simulation has been trace-driven based of cache logs taken from DEC servers [8]. Table 1 summarizes the characteristics of the employed traces and the simulated caching subsystem.

Table 1. List of Simulation Parameters

Number of Clients	10,366
Number of Requests	4,431,200
Replacement Algorithm	LRU
Cache Size	128MB, 256MB, 512MB, 1GB, 2.5GB, 5GB, 10GB

The metrics recorded throughout the simulation were the following: 1) the observed total cache hit rate (*HR*), 2) the hit rate experienced by client *i* (HR_i), 3) the coefficient of variation of the HR_is, 4) the number of invocations to the LRU cache replacement mechanism, and 5) the mean usage of the cache space. The cache hit rate (*HR*) is calculated as

$$HR = \frac{\text{Number of Hits}}{\text{Number of Requests}}$$

while the hit rate HR_i for client *i* is calculated as follows:

$$HR_i = \frac{\text{Number of Hits of Client } i}{\text{Number of Requests of Client } i}$$

We have adopted the *coefficient of variation* of the HR_i measurements as an indicator of the fairness achieved by the caching scheme. A high value of this fairness criterion (FC) means that different clients do not enjoy the same benefits from caching and some monopolize the disk space. Conversely, a low FC value implies that the behavior experienced by the majority of clients is almost identical and all have enjoyed an almost equal hit ratio. Specifically, the fairness metric is defined as

$$FC = \frac{1}{\overline{H}} \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(HR_i - \overline{H} \right)^2} \qquad (10)$$

where $\overline{H}$ denotes the mean client hit rate and is calculated as follows:

$$\overline{H} = \frac{1}{N} \sum_{i=1}^{N} HR_i$$

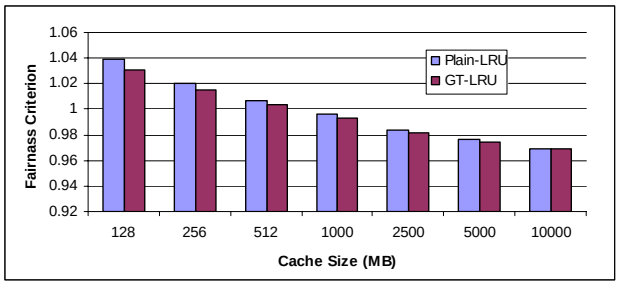

Figure 5. Fairness Criterion

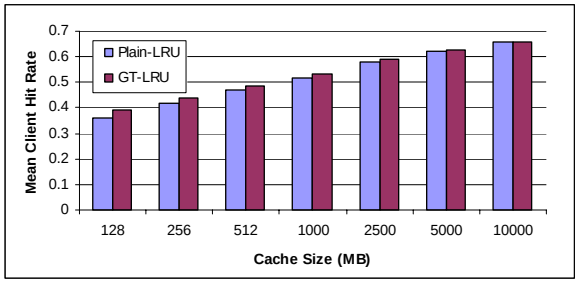

Figure 6. Mean Client Hit Rate

As shown in Fig. 5, the game theoretic mechanism achieves lower FC values. Moreover, as shown in Fig. 6, the proposed mechanism yields an increase in the mean client hit rate $\overline{H}$ with respect to the plain-LRU scheme. Such increase may be attributed to the fact that, in the game theoretic case, "highly aggressive" clients, who are typically few, cease caching, according to the proposed mechanism, thus, do not monopolize the cache disk space. As a consequence, "less aggressive" clients, who are the majority, take benefit of the available disk space and achieve an increased hit rate. However, as anticipated, the overall cache hit rate (*HR*) is decreased, as shown in Fig. 7.

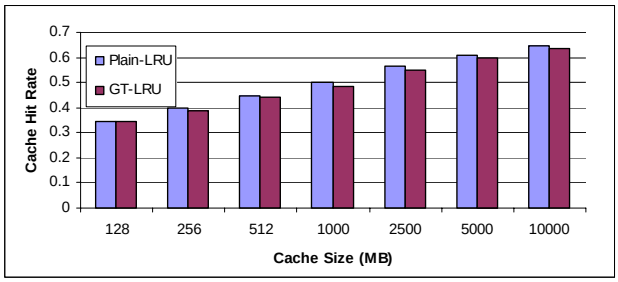

Figure 7. Cache hit rate

The proposed mechanism managed to drastically reduce the number of LRU invocations in the cache, as shown in Fig. 8. Specifically, the number of replacements in the game theoretic scenario was significantly reduced with respect to the unregulated case, especially for small cache sizes. In every invocation, the replacement scheme removed the least recently used items of the cache to free the 20% of the allocated disk space.

We have also quantified how the mean cache disk space use (a time average) is influenced by the application of the game theoretic scheme. Fig. 9 shows the relevant findings. In general, the level of mean cache disk space use is lower in the game theoretic case, than in the unregulated scenario. This is anticipated, since clients do not cache objects deliberately and the cache disk is filled at a slower rate.

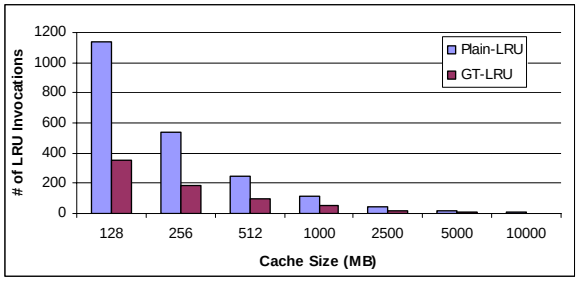

Figure 8. Number of LRU invocations

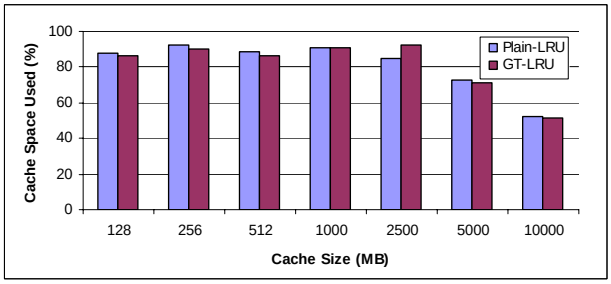

Figure 9. Mean Cache Disk Space Use

7. Prior Related Work

We are aware of only a few very recent works on game-theoretic aspects of web content caching and replication. The most relevant to our work is the one in [13], where the contention between different clients that compete for the storage capacity of a single cache server has been studied, and modeled as a non-cooperative game. However, this work did not provide an analytical model of the proposed framework, and the simulations were based on synthetic traces. Other, less relevant, works deal with distributed selfish replication (DSR). DSR is defined as the replication of resources by server nodes that act selfishly. This problem has been studied as a non-cooperative game in [16]. More recently, in [14], equilibrium object placement strategies were derived for the DSR problem, and two distributed algorithms have been proposed. Another related work on market-based resource allocation in content delivery networks is reported in [15]. The focus in [15] is on a three-part market game between content providers, distributors, and consumers.

8. Conclusions and Future Work

In the context of Web caching, clients typically reserve more disk space in order to improve their cache performance (hit rate). As the cache disk space constitutes a finite resource, a social interaction problem is formulated. In this paper, we have formulated the problem as a non-cooperative game, and the existence of a NE has been investigated. Clients compete with each other trying to selfishly maximize their utility function until the NE is reached. We propose a utility function consisting of profit and cost components. Clients have to determine a rational course of interaction taking into account the performance advantages and associated resource retrieval and storage costs. The cost component is dependent upon the strategies that different clients assume. An extensive simulation of the game theoretic mechanism has been performed. Our findings indicate noteworthy improvement in the adopted fairness criterion metric. The performance seen by different clients is comparable and more predictable compared to the unregulated setting. At the game theoretic scenario, the number of cache replacement operations is drastically reduced and the mean client hit rate is higher, however, the overall cache hit rate is marginally degraded. Our future work in this area includes the assessment of the impact of similar game theoretic schemes in cooperative caching (e.g., in caching hierarchies) as well as web content pre-fetching.

9. Acknowledgements

The first author would like to thank the Alexander S. Onassis Public Benefit foundation for its financial support.

10. References

[1] B. M. Duska, D. Marwood and M. J. Feeley, “The Measured Access Characteristics of WWW Client Proxy Caches”, Proc. USENIX Symposium on Internet Technologies and Systems, December 1997.

[2] A. Bovopoulos and A. Lazar, “Decentralized algorithms for optimal flow control”, Proc. Allerton Conference on Communications, Control, and Computing, 1987, pp. 979-988.

[3] D. Fudenberg and J. Tirole, “Game Theory”, MIT Press, Cambridge (MA), 1991

[4] J. B. Rosen, “Existence and Uniqueness of Equilibrium Points for Concave N-Person Games”, Econometrica, Vol.33, No.3, 1965.

[5] S. H. Rhee and T. Konstantopoulos, “A Decentralized Model for Virtual Path Capacity Allocation”, Proc. INFOCOM 1999.

[6] D. P. Bertsekas and J. N. Tsitsiklis, “Parallel and Distributed Computation: Numerical Methods”, Athena Scientific, Belmont, MA, 1997.

[7] A. Lazar, A. Orda and D. Pendarakis, “Virtual Path Bandwidth Allocation in Multiuser Networks”, IEEE/ACM Transactions on Networking**,** Volume 5, Issue 6, December 1997, pp. 861 – 871

[8] Digital Equipment Corporation, Digital’s Web Proxy Traces. ftp://ftp.digital.com/pub/DEC/traces, August–September 1996.

[9] M. Rabinovich and O. Spatscheck, “Web caching and replication”, Addison Wesley, 2002.

[10] C. Roadknight and I. Marshall, “Variations in cache behavior”, Proc. 7th International WWW Conference (WWW7), Brisbane, Australia, April 1998.

[11] H. Nielsen, P. Leach and S. Lawrence, “An HTTP Extension Framework”, IETF Network Working Group, RFC 2774, February 2000.

[12] P. Konstanty and M. Kozipski, “Web Cache charging policies”, position paper in NLANR Web Caching Workshop, Boulder, USA, 1997.

[13] S. Hadjiefthymiades, Y. Georgiadis and L. Merakos, “A Game Theoretic Approach to Web Caching”, Proc. 3rd International IFIP-TC6 Networking Conference, Athens, Greece, 2004.

[14] N. Laoutaris, O. Telelis, V. Zissimopoulos and I. Stavrakakis, “Distributed Selfish Replication”, accepted in IEEE Transactions on Parallel and Distributed Systems, 2005.

[15] Ö. Erçetin and L. Tassiulas, “Market-Based Resource Allocation for Content Delivery in the Internet”, IEEE Transactions on Computers 52(12), pp. 1573-1585, 2003.

[16] B. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. Papadimitriou and J. Kubiatowicz, “Selfish caching in distributed systems: a game-theoretic analysis”, Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing, Canada, 2004.