STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 3, 2011

DYNAMICS WITHIN A LAN DISTRIBUTED VIDEO PROXY-CACHE

CLAUDIU COBÂRZAN AND DIANA MAN

Abstract. We introduce a fuzzy clustering approach that uses a well known algorithm to partition the clients in a LAN. This is done in the context of a distributed video proxy-cache that is able to vary the number of active nodes based on LAN conditions and client behavior.

1. Introduction

Video proxy-caching within LANs is an often deployed solution that tries to tackle some of the problems due to increasing demand for multimedia and especially audio-video content from the Internet.

We defined a distributed system that aims at providing better support for clients within a LAN by varying the number of active nodes depending on a number of conditions.

In the following we will make an overview of the system dynamics and introduce a new approach to determining the node that should host a new proxy-cache in case another one is needed. The new approach uses fuzzy clustering to cluster the existing nodes in the LAN with the cluster centers hosting the running proxy-caches. The number of clusters equals the number of running proxy-caches minus one, since we consider that the systems starts with one active proxy-cache.

2. Overview of the Proxy-Caching System

In [3] we introduced a video-proxy-caching system that starts with a single active node, but can add and then remove nodes from the system as necessary. Adding a new node to the proxy-caching system is done when new computing and/or storage resources are needed to service client requests. Removing nodes from the system takes place when the request volume drops and fewer nodes can deal with the existing clients.

Received by the editors: May 15, 2011.

2000 *Mathematics Subject Classification.* 68-06, 68M14.

1998 *CR Categories and Descriptors.* H.3.4 [**Information Storage and Retrieval**]: Systems and Software – Distributed systems.

Key words and phrases. video proxy-cache, fuzzy, fuzzy clustering.

2.1. **System Dynamics.**

The system is implemented by using two entity types: `dispatchers` and `daemons`. The `dispatchers` run on every proxy-cache node and have coordinator responsibilities: they contact their siblings or origin servers in order to solve requests they can't service from the local cache. Also, they select the node that will host a new proxy-cache, when such an operation is required. The `daemons` run within a LAN on the nodes that can become proxy-caches (potentially on every node). They also act as on-site proxies by selecting one `dispatcher` from the list of active ones to which a client request is forwarded.

The conditions that trigger a *split* operation (that adds a new node), as well as a *hibernate* or *shut down* operation (that remove a node) are detailed in [3] and in [4]. When performing a *split* operation, a `dispatcher` selects the "best" `daemon` to host the proxy-Cathe code and possibly some of the data from the local cache. Leaving the system is done by a *shut down* operation that stops the proxy-cache and its `dispatcher` and discards any locally stored data. A proxy-cache can enter a dormant state in which it only services requests for objects it already stores by performing a *hibernate* operation. If a *split* operation has to be performed and there are hibernating nodes, one of those nodes will be reactivated.

Deciding which nodes will be shut down, put in hibernation or reactivated is done by their *rank* value according to the conditions in [3] and [4]. The ranking mechanism was introduced in [4] and refined in [2] at node level. The idea behind it is to rank the active or hibernating proxy-caches by the volume of served data. In [2] we propose differentiating between data sources - local cache, siblings or origin servers - so that the largest rank is assigned to the node(s) serving the largest amount of data from within the local cache when reported to the amount of data served from sibling caches and remote servers.

2.2. **Cache Operations.**

All cache specific operations (add a new object, discard, move or replicate an object) are done according to the *utility* of the objects. The *utility* value is computed when an object first enters the cache, but varies over time according to several characteristics: size, number of requests, time of the last request etc. The considered characteristics have different weights. Those weights are either static (fixed for the life time duration of the proxy-cache) or are dynamically generated (the values may change during the life time of the proxy-cache). In [9] and [5] we proposed using genetic algorithm when determining the weighting values so that the system's efficiency (measured with *byte-hit-rate* as metric) is maximized. The byte-hit-rate is computed as ratio between the data volume served from within the cache and the data amount served from the cache, siblings and servers (total amount of served data).

A video object $o_{ij} \in LC_i$ held in the cache $P_i \in P$ ($1 \leq j \leq q$, q - the number of cached objects at node i) is defined in [4] as:

$$o_{ij} = (size(o_{ij}), duration(o_{ij}), bitRate(o_{ij}), qualityValue(o_{ij}), TLA, HC, COST) \tag{1}$$

(LC_i stands for the contents of the local cache while P represents the set of active proxies).

The proxy-cache holds information on the size, duration, encoding bit rate and quality value of the video, where the quality value (a real number between 0 and 1) is the measure of the object's quality (based on characteristics of the video object like resolution, color information etc.) to different clients.

Also, three vectors are used to hold additional data:

- the moments in time when the object was last requested - the *TLA (Time of Last Access)* vector
  (e.g. $timeLastAccess(o_{1,3})_{N_2}$ represents the moment the 3^{rd} object cached at node P_1 has been last requested from the node N_2);
- the number of requests for the object - the *HC (Hit Count)* vector
  (e.g. $hitCount(o_{1,3})_{N_2}$ represents the number of times the 3^{rd} object cached at node P_1 has been requested from node N_2);
- the cost of streaming the object from the cache to the requesting client - the $COST$ vector
  (e.g. $cost(o_{1,3})_{N_2}$ represents the cost of streaming the 3^{rd} object in the cache at node P_1 to the node N_2. It is computed as follows:

$$cost(o_{1,3})_{N_2} = \frac{\alpha_{P_1,N_2}}{bitRate(o_{1,3})} duration(o_{1,3}) \tag{2}$$

  meaning "the cost of streaming the 3^{rd} object from the cache at node P_1 to the node N_2 equals the amount of bandwidth needed to stream that particular object" - α_{P_1,N_2} denotes the amount of bandwidth available between node P_1 and node N_2).

3. A New Approach to System Expansion

We propose a new approach for determining the node that will host a new proxy-cache. This approach is based on fuzzy clustering.

Clustering is the division of a data set into subsets (clusters) such that, similar objects belong to the same cluster and dissimilar objects to different clusters. In real applications, there is no obvious boundary between clusters so that, fuzzy clustering is often better. In fuzzy clustering the object can belong to more than one cluster so, it has a degree of belonging to clusters, as in fuzzy logic.

A hard clustering algorithm allocates each object to a single cluster. Thus, traditional clustering divides data objects in partitions. *Fuzzy clustering* extends this notion to associate each data object with every cluster using a membership function. The output of the hard clustering algorithm is a partition, whereas the one of fuzzy algorithm is a clustering. In fuzzy clustering each cluster is a fuzzy set of all the patterns. In general the performance of fuzzy clustering algorithms is superior to that of the corresponding hard algorithms. Fuzzy clustering has proved successful in many relevant applications from real world.

The most popular fuzzy clustering algorithm is *FCM - Fuzzy c-Means* [6, 1]. It is a data clustering technique in which each data point belongs to a cluster to some degree that is specified by a membership grade. The algorithm is also known as *Fuzzy ISODATA* or *Fuzzy K-Means*.

Given a set $X = \{x_1, x_2, ..., x_n\} \subset \Re^p$ of sample data, the aim of the algorithm is to determine the prototypes (cluster centers) in such a way that the objective function is minimized.

In our case the set X is composed by the set of nodes (N).

The objective function $J : NX[1, \infty) \to R$ is:

$$J(c, q) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c}(\sum_{k=1}^{n} u_{ik}^q d_{ik}^2)), q \in [1, \infty) \tag{3}$$

subject to:

$$\sum_{k=1}^{n} u_{ik} > 0, \forall i \in \{1, 2...c\}, \sum_{i=1}^{c} u_{ik} = 1, \forall k \tag{4}$$

where u_{ik} stands for the membership degree of datum x_k to cluster i, d_{ik} is the distance of datum x_k to cluster i, represented by the prototype p_i and c is the number of clusters. The parameter q is a weighting exponent (fuzziness exponent). Usually $q = 2$ is chosen. At $q = 1$, *FCM* collapses to *HCM algorithm* [7].

The first constraint guarantees that no cluster is empty and the second condition ensures that the sum of the membership degrees for each datum equals 1.

Also, this constraint corresponds to a normalization of the membership per datum. As a consequence of both conditions no cluster can contain the full membership of all data points. Thus, the membership degrees for a given datum formally resemble the probabilities of its being a member of the corresponding cluster.

The output of the *FCM* algorithm is not a partition, thus: $C_i \bigcap C_j \neq \emptyset, i \neq j$.

There are two necessary conditions for J to reach a minimum:

$$p_i = \frac{\sum_{k=1}^{n} u_{ik}^q x_k}{\sum_{k=1}^{n} u_{ik}^q} \tag{5}$$

$$u_{ik} = \frac{(\frac{1}{d_{ik}})^{1/(q-1)}}{\sum_{j=1}^{c} (\frac{1}{d_{jk}})^{1/(q-1)}} \tag{6}$$

where d_{ik} is the *distance* between object x_k and the center of cluster C_i [8].
`The FCM Algorithm:`

```
1. Initialize the membership matrix U with random values between
0 and 1 within the constraints of (2).
2. Calculate c cluster centers $p_i, i=1..c$ using (3).
3. Compute the objective function according to (1). Stop if
either it is below a certain threshold level or its improvement
over the previous iteration is below a certain tolerance.
4. Compute a new U using (4).
5. Go to step 2.
```

The *FCM* algorithm assumes that the number of clusters c is known. In real cases, this parameter is not known and must be determined. The traditional approach to determining c is to evaluate a certain global validity measure of the c-partition for a range of c values and then pick the value of c that optimizes the validity measure. An alternative is to perform progressive clustering where clustering is initially performed with an over specified number of clusters. After convergence, bad clusters are eliminated, compatible clusters are merged and good clusters are identified [10].

The system we defined starts with one proxy-cache that will be permanently active but it can add proxy-caching nodes whenever necessary. That is why we set the number of clusters to 1 when the first additional proxy-cache is added. In this situation we have to determine the center of the cluster formed by all the nodes in the LAN (the set N) which will be the node hosting the second proxy-cache. However there is a constraint: in order to host the new proxy-cache, the center has to have a running `daemon`. Whenever there are more than one active proxy-cache, the number of clusters equals the number of active proxy-caches minus one, and the nodes hosting them are the centers of those clusters (if the designated centers have running `daemons`). Whenever a *split* operation is performed, the nodes in the LAN are re-clustered and the number of clusters increases by one.

We will consider the *distance* between the node x_k and the center of cluster C_i the report between the total cost of streaming the stored all objects and the total number of requests for the objects stored on the node. This report must be minimized.

4. Conclusions and Future Work

We introduced a new method of determining the nodes that will host a new proxy-cache in a dynamic distributed video proxy-caching system. This methods clusters the existing clients based on the cost of delivering objects to those clients as well as client activity (number of requests). As future work to intend to refine the conditions for determining those nodes by also considering the time when the client requests were made and also to develop algorithms for moving data between the active nodes (data move, data replicate) that consider client clusters.

References

1. James C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*, Kluwer Academic Publishers, Norwell MA, USA, 1981.
2. Caudiu Cobârzan, *Node Ranking in a Dinamic Distributed Video Proxy-Caching System*, KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEEPT2007, 6-8 June 2007, Cluj-Napoca, Romania (Str. Haşdeu nr. 45, 400371, Cluj-Napoca, Romania), Presa Universitară Clujeană/Cluj University Press, 2007, pp. 298–306.
3. Claudiu Cobârzan, *Dynamic Proxy-Cache Multiplication inside LANs*, Euro-Par 2005, Lecture Notes in Computer Science, vol. 3648, Springer, 2005, pp. 890–900.
4. Claudiu Cobârzan and László Böszörményi, *Further Developments of a Dynamic Distributed Video Proxy-Cache System*, Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007), IEEE Computer Society, 2007, pp. 349–357.
5. Claudiu Cobârzan, Alin Mihăilă, and Cristina Mihăilă, *Dynamics of a Utility based Distributed Video Proxy-Cache*, 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2008), September 26-29, 2008, Timisoara, Romania, 2008, (accepted for publication in IEEE post-proceedings).
6. J.C. Dunn, *A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters*, Journal of Cybernetics **3** (1973), no. 3, 32–57.
7. A.K. Jain, M.N. Murty, and P.J. Flynn, *Data clustering: A review*, ACM Computing Surveys **31** (1999), no. 3, 264–323.
8. Jan Jantzen, *Neurofuzzy modelling*, Technical Report 98H874 (nfmod), Technical University of Denmark, 1998.
9. Cristina Mihăilă and Claudiu Cobârzan, *Evolutionary approach for multimedia caching*, 19th International Workshop on Database and Expert Systems Applications (DEXA 2008), 1-5 September 2008, Turin, Italy, IEEE Computer Society, 2008, pp. 531–536.
10. Horia F. Pop, *Data analysis with fuzzy sets: a short survey*, Studia Universitatis Babes-Bolyai, Series Informatica **XLIX** (2004), no. 2, 111–122.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania
E-mail address: {claudiu,mandiana}@cs.ubbcluj.ro