An Efficient Framework for Managing Consistency in CDN

P.Venketesh [1] S.N.Sivanandam [2] S.Manigandan [3] A.Chitra [4]

Abstract

Content Distribution Networks (CDN) plays a major role in reducing client latency and minimizing network workload. To disseminate data effectively to end users spread geographically, Consistency has to be maintained between origin server, surrogate servers and intermediate proxies. In this paper, we propose a novel approach that provides scalable and flexible solution for maintaining consistency. The model uses concept of Cooperative Consistency along with mechanism called Object Lifetime to achieve strong/weak consistency depending on the nature of object accessed. The proposed model will reduce message and state space overhead at both server and proxy significantly. To send server notifications to proxies, we use application-level multicast that enhances performance and applies in a scalable manner to content distribution network.

Key words :Content Distribution Network, Data Dissemination, Object Lifetime, Consistency, Caching, Propagation, Invalidation.

1. INTRODUCTION

The World Wide Web and Internet has witnessed enormous growth for more than a decade and it demands efficient mechanism to disseminate data to end users. Content Distribution Network (CDN) is considered a viable option; it minimizes client latency, reduces server and network load significantly. CDN comprises origin server, surrogate servers and intermediate proxies.

Consistency maintenance among these components is very essential for delivering current data to end-users. Different objects require different level of consistency guarantees based on their characteristics and user preferences. Consistency mechanism indicates four different levels that can be considered for implementation: *Strong Consistency*, *Delta Consistency*, *Weak Consistency* and *Mutual Consistency*.

Consistency model to be employed by CDN should satisfy two important factors: *Scalability* and *Flexibility*. Methods proposed for stand-alone proxies do not scale well for large number of proxies available in CDN. The original leases approach has several drawbacks when applied to busy servers. The problem of finding exact lease duration is a major concern and it causes significant computation overhead for server. In this paper, we propose a model that uses concept of *Cooperative Consistency* along with mechanism called *Object Lifetime*. Several Proxies cooperate with each other to achieve desired level of consistency.

In this model, server notifications will be sent to cooperating proxies until object is removed from corresponding proxy. It ignores computation of leases and minimizes server workload. By employing Δ consistency, the model provides flexible way of specifying consistency guarantee to be provided for

[1,3]Research Scholars, Department of CSE, PSG College of Technology, Coimbatore, India.

[2,4]Professors, Department of CSE, PSG College of Technology, Coimbatore, India.

Email:venkip_ms@yahoo.co.in[1], manigandan_me@yahoo.co.in[3]

particular object. To disseminate server notifications, we use application-level multicast, which enhances server performance. Request / Response messages use existing HTTP/1.1 Protocol specifications with slight modification to meet our requirements. The proposed model will reduce number of messages exchanged between server and proxy; minimizes server state space to a considerable extent.

The rest of this paper is organized as follows. Section 2 provides related work carried out in maintaining consistency. Section 3 explains the proposed system model that provides scalable mechanism for consistency management. Section 4 discusses advantages and design issues of proposed approach. Section 5 concludes the paper.

2. RELATED WORK

Consistency for CDN caches is implemented by selecting appropriate *consistency models* that uses various *consistency policies* and *content distribution mechanisms G.Pierre et al* [1]. Consistency model is basically a contract between content delivery system and its clients that dictates the consistency-related properties of the content. Consistency policy defines how, when, and to which object replicas the content distribution mechanisms are applied. Replica servers exchange replica updates using content distribution mechanisms. *A.Iyengar et al* [2] broadly categorizes Consistency provisioning as: Server Driven, Client Driven and Explicit mechanisms.

H.Yu et al [3] proposes architecture that uses caching hierarchy and application level multicast routing to convey invalidations. It forms multicast group among caches and sends heartbeat message to each other. *M.Dahlin et al* [4] proposes Web Cache Invalidation Protocol (WCIP) for propagating server invalidations using application-level multicast while providing delta consistency. *Z.Fei* [5] suggests a novel hybrid approach for maintaining strong consistency between origin server and surrogate servers. The origin server makes the decision of using either propagation or invalidation method for each document based on the statistics about the update frequency at the origin server and the request rates collected by replicas

Cache consistency is achieved through a protocol called WCDP [6] (Web Content Distribution protocol). It is an Invalidation and Update protocol with different consistency levels like *Strong Consistency*, *Delta Consistency*, *Weak consistency* and *explicit consistency*. *V.Duvvuri et al* [7] suggests Leases as a mechanism to provide consistency in CDN. It broadly categorizes leases into three groups: *age-based, renewal-frequency-based and load based ones*. *J.Yin et al* [8] proposed mechanism called Volume Leases, where each lease represents multiple objects cached by stand-alone proxy. *A.Ninan et al* [9, 10] proposed a scalable and flexible technique called *Cooperative consistency* along with mechanism of *Cooperative leases*. It uses "ΔConsistency semantics and single lease for multiple proxies. *S.Sivasubramanian et al* [11] proposed a system that guarantees *strong consistency* for web applications with high *scalability*. It uses unique mechanism of Partial Replication, where data units are replicated only to servers that access them often.

In order to provide Strong Consistency for web caches a protocol called BTC (Basis Token Consistency) was proposed by *Adam D.Bradley & Azer Bestavros* [12]. *J.Yin et al* [13] proposed an approach that uses hierarchical structure for server-driven cache consistency. *Haifeng Yu & Amin Vahdat* [14] proposes continuous consistency model, which explores semantic space between strong and optimistic consistency guarantees.

In order to capture consistency spectrum it defines three metrics: *Numerical Error*, *Order Error* and *Staleness*. The distributed object consistency protocol (DOCP) [15] proposes extensions to the current HTTP cache control mechanism for providing consistency guarantees. DOCP uses publish and subscribe mechanism along with server invalidations to provide consistency guarantees. Slave proxies use an optimistic discovery mechanism to subscribe for content published by Master proxies.

3. CONSISTENCY MODEL

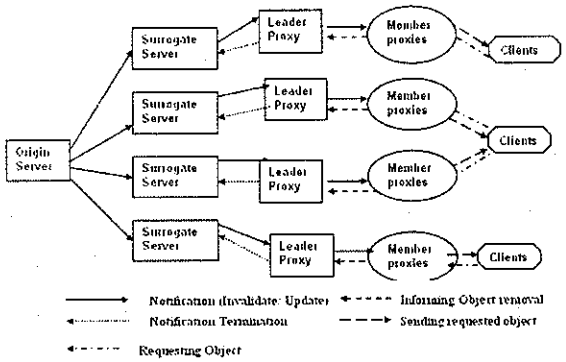


Fig1: Consistency Model System Architecture

Content Distribution network comprises Origin server, Surrogate Servers, intermediate proxies and clients as depicted in fig1. Web proxies present in CDN will be logically organized in to non-overlapping regions for each object to minimize overhead in maintaining consistency. Each region will select *Leader* proxy that is responsible for communicating with surrogate servers. Between Origin server and Surrogate servers, the system should maintain strong consistency because surrogate server contains exact replica of data present in origin server. Consistency level applied between Surrogate servers and intermediate proxies will depend on object characteristics and user preferences. To achieve scalable way of specifying the consistency guarantee to be provided for particular object, we use concept called as Δ- consistency. The value of Δdetermines the nature of consistency provided for an object; larger the value of Δweaker the consistency guarantee provided (Δ=0, indicates strong consistency).

In this model, we introduce concept called as *Object Lifetime*- amount of time object resides in cache before being purged. Modifications done to object present at server needs to be notified to proxies for maintaining current version of object. Server notification to proxies can be either Invalidation message or update of new object; choice depends on various factors like: Modification rate at server, request frequency of object in proxy, size of object. Notification from server will be continued till corresponding object is removed from proxy. Proxies in CDN cooperate together to achieve consistency- called as *Cooperative Consistency* (one Leader proxy, many member proxies per region). Server notifications will be sent only to leader proxy, it minimizes amount of state to be maintained at server and number of messages to be sent. Leader proxy is responsible for propagating notifications to its member proxies.

Leader Selection and logical grouping of proxies need to be performed for every object. Method used for leader selection should be such that it achieves load balancing; i.e. all proxies get equal opportunity to be a leader for different object. Each Leader needs to maintain *Membership* list that holds list of proxies interested in particular object. Similarly, member proxies will maintain a list that indicates *Leader* for particular object and also information about other proxies that holds the object. To maintain mapping between Leader proxy and member proxies mechanisms like consistent hashing [16], hint caches [17] and bloom filters [18] are considered.

3.1 First Time Requests

When client requests an object for the first time it is sent to a proxy within a region as shown in Fig 2. Before proxy sends the request to server, it initiates leader selection algorithm for selecting *Leader* of corresponding region; different object will have different leader. Since proxy may not hold the object initially, it directs the request to either origin server or surrogate server. It sends leader information and notification rate-Δ(optional) along with the request to server. HTTP/1.1 Protocol used to generate requests and responses. Server responds by providing the requested object along with Object_ID to requested proxy and also to Leader if it is different from the proxy that requested it. The usage of Object_ID in subsequent transaction will reduce the amount of data to be sent between server and proxy. *Leader* will broadcast message to all proxies in the region informing about its availability. Server will propagate notification only to leader proxy and it is responsible for managing objects in member proxies.

Request: (Proxy to Server)

{URL, L, Δ}

Δ- Notification rate (optional)

L – Leader of Region

URL- link for the object

Response: (Server to Proxy)

{O, O_ID, L, Δ} [Assuming only one region considered]

{O, O_ID, G, L, Δ} [Assuming more than one region for CDN]

O= Object, O_ID= Object ID,

G= Proxy region, L= Leader,

Δ= Notification rate

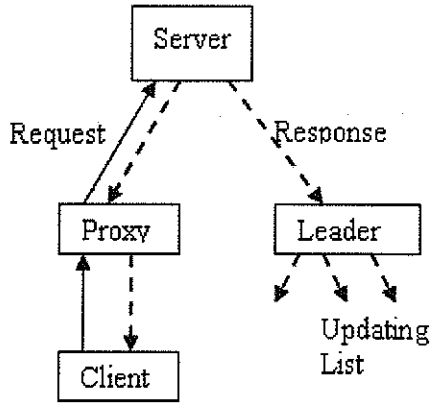


Fig. 2: First Time Requests

3.2 Subsequent Requests

When client makes request for the same object, proxy first checks in its local cache. If it is Local hit, requested object will be send to client from proxy as indicated in Fig 3. If it is a miss, proxy will send the request to either server or to other proxies within region. Leader has to update its membership list every time proxy gets object from server, so that the corresponding proxy will receive notification. The member proxy may use Object_ID sent to it from server or it can use its own Local_Object_ID. If it uses different identifier, then mapping has to be provided between Object_ID from server and Local_Object_ID of member proxy.

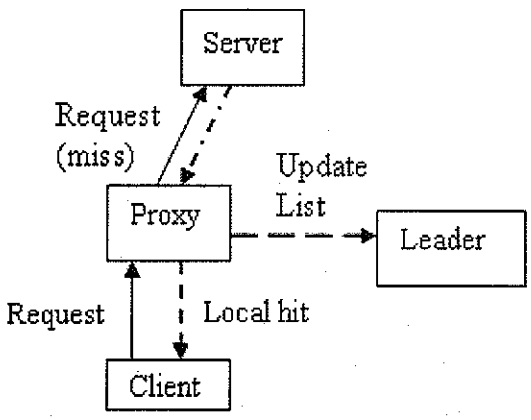


Fig. 3: Subsequent Requests

3.3 Object Updates

Whenever there is modification to object at server, it will send notifications to proxies to maintain consistent view of data. The notifications sent from server can be either invalidation message or update message containing new version of object. Depending on the notification rate Δ, server propagates the notification messages only to leader proxies in order to minimize amou nt of messages to be transmitted. Leader proxy is responsible for sending notifications to its member proxies to maintain consistency, based on the membership list it maintains. Only leader proxy will interact with the server and it is responsible for terminating notifications. Between Server and Leader proxy, we use Unicast mechanism to send messages. Leader will use Application-level multicast mechanism to propagate notifications to its member proxies.

If Invalidation message need to be sent from server to leader proxy and from leader proxy to member proxies, message format used is

{Object_ID, invalidate}

If Update message is to be sent, then message format used is

{Object, update}

(Or)

{Object, Object_ID, update}

Object here corresponds to new version of the object sent for maintaining consistency.

3.4 Notification Termination

Leader Proxy is responsible for terminating the notification from server. Whenever object is removed from member proxy, it will intimate to *Leader* and it will not send notifications to corresponding member proxy. When membership list maintained at *Leader* becomes empty, it will send termination message to server and intimation message to member proxies saying it will be no longer leader for corresponding region.

Whenever a particular object gets hit in proxy, it will be placed at the head of Queue that is maintained for easy maintenance of objects. When an object does not get any hit, it will move down the queue, time taken to transfer down the queue depend on the input rate of requests coming to corresponding proxy. Object lifetime depends on the number of hits a particular object receives while it is in the corresponding proxy. The popularity of an object depends on how long it resides in cache and number of hits it receives. An object will be purged from cache, if it does not get any hit or number of hits it receives is minimal. The replacement policy used for object removal depends on various factors like: Object Hit Rate, Byte Hit Rate and QoS desired.

Independent of Replacement policy used, the object that gets low hit will be moved towards the tail of queue, and its probability as a victim for removal from cache is high. To notify that object has been removed from proxy, we use Object_ID instead of using Object; it reduces amount of data to be sent. The message format used by member proxy to inform purging of object to Leader will be:

{Object_ID, Purged}

where Purged is a Boolean value (True/ False)

Whenever leader proxy gets this message, it removes particular proxy information from its membership list; notifications for that proxy will be terminated.

4. DESIGN ISSUES

The model requires cooperation from both proxy and server for maintaining consistency. Server is responsible for sending notifications to leader proxies. Member Proxy is responsible for tracking when object gets purged from cache and inform to leader proxy, so that it will not get any notification for particular object. Advantages of this model include:

- Minimal Server Overhead; State space maintained is significantly reduced.
- No Leases considered, so computation is reduced at server.
- Need not consider any renewal methods for getting notifications from server.
- System is highly flexible and scalable, can be applied to large CDN.
- No Unnecessary notifications; whenever object is removed from proxy, it will not get any message.

Proxy need not renew duration time from server for getting notifications, as long as object resides in cache. It minimizes client latency when it makes request for the object.

Each proxy maintains mapping between object and its corresponding leader; it also maintains information about other proxies holding the same object. The information will be maintained till leader is relieved from its activity, even though object is removed from proxy. This mechanism will help the member proxy to contact the same leader, if client makes request to the same object after it is removed. Proxy will select a new leader, only when current leader sends message saying that it is relieved from the job.

If Object is popular and resides in corresponding proxy for a long duration, Server will keep on sending notification to *Leader*; which in turn sends to member proxy without knowing whether corresponding proxy is alive or not. To check the status of Server, Leader Proxy and Member Proxies, we use mechanism called as 'HeartBeat'. At periodic intervals, the system will be sending heartbeat messages to keep it in working condition. Based on certain factors, interval value can be fixed by either client or content provider or by both. The information obtained using this mechanism, helps to prevent unnecessary transfer of messages, removing information from server/ proxy that is invalid.

If cache space maintained at proxy is not full, then object may not be removed from it by the corresponding replacement policy. This may lead to endless notification from Server. To overcome this situation, time duration can be set, such that if object does not get any hit within fixed time, then it can be removed from cache or marked invalid. The notification rate-•can be calculated based on server or network workload.

4.1 Test Results

The proposed model is tested by considering an origin server, surrogate server, set of proxies and clients. The model is compared with existing leases approach to indicate the enhanced performance provided by our system. In fig 4, we have shown the number of control messages exchanged during different time periods. For renewal times (10,15,20,25 min) the leases approach requires more control messages compared to our proposed model.

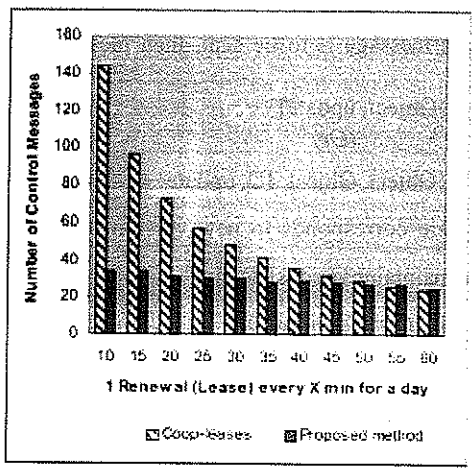


Fig 4: Analysis of Test Results

The new approach achieves more or less equal performance to leases approach when the renewal time is more. The approach shows significant reduction in message transfer when object resides in proxy for long duration.

5. CONCLUSION

In this paper, we have proposed a new model that uses *Object Lifetime* along with mechanism called *Cooperative Consistency*. The model will meet requirements such as scalability and flexibility in maintaining consistency. It significantly reduces the overhead experienced in previous methods. It can be applied effectively for both static and dynamic data available in CDN. Leader selection method should be such that it achieves efficient load balancing. The model supports different levels of consistency, uses existing HTTP /1.1 protocol for sending invalidates and updates.

References

[1] S. Sivasubramanian, M.Szymaniak, G.Pierre, Marteen Van Steen, "*Web Replica Hosting System Design*", Internal Report IR-CS-001, Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, Revised May 2004

[2] A.Iyengar, E.Nahum, A.Shaikh, R.Tewari "*Enhancing Web Performance*" Proceedings of the IFIP 17th World Computer Congress – TC6 Stream on Communication Systems: the State of the Art, pages 95-126, Year: 2002

[3] H.Yu, L.Breslau, and S.Shenker. "*A Scalable Web Cache Consistency Architecture*" In *Proceedings of the ACM SIGCOMM '99*, Boston, MA, Sep.1999

[4] D.Li, P.Cao and M.Dahlin "*WCIP: Web Cache Invalidation Protocol*", *IETF Internet Draft*, November 2000

[5] Z.Fei. "*A Novel Approach to Managing Consistency in Content Distribution Networks*" In *Proceedings of the 6th Workshop on Web Caching and Content Distribution, Boston,* MA, June 2001.

[6] R. Tewari, T. Niranajan, and S. Ramamurthy, "*WCDP: Web content distribution protocol*" *IETF Internet Draft*, March 2002.

[7] V. Duvvuri, P.Shenoy, and R.Tewari, "*Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web*", In *Proceedings of the IEEE Infocom `00, TelAviv, Israel*, March 2000.

[8] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "*Volume Leases for Consistency in Large-Scale Systems*", IEEE Transactions on Knowledge and Data Engineering, January 1999.

[9] A.Ninan. "*Maintaining Cache Consistency in Content Distribution Networks*" *Master's Thesis*, Department of Computer Science, Univ. of Massachusetts, June 2001

[10] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari, "*Cooperative Leases: Scalable consistency maintenance in content distribution networks,*" *in WWW2002*, (Honolulu, Hawaii), May 2002.

[11] S.Sivasubramanian, G. Pierre, Maarten van Steen, "*Scalable Strong Consistency for Web Applications*", *In Proceedings of 11th ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[12] Adam D. Bradley and Azer Bestavros, "*Basis Token Consistency: Supporting Strong Web Cache Consistency*", *Global Internet 2002 (GI 2002)*, Taipei, Taiwan, 2002.

[13] J.Yin, L.Alvisi, Mike Dahlin, Calvin Lin, "*Hierarchical Cache Consistency in WAN*", In Proceedings of the Usenix Symposium on Internet Technologies (USITS '99), Boulder, CO, October 1999.

[14] Haifeng Yu and Amin Vahdat, "*Design and valuation of a Continuous Consistency Model for Replicated Services*", In Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, California, October 2000.

Fractal compression assumes that every image is composed of smaller images just like them. The compression technique tries to find as many of these relationships in an image and then describe them with mathematical formulas. This is done within regions of an image called domain regions. These domain regions are determined by using techniques such as frequency analysis, edge detection, and texture-variation analysis.

In conventional fractal image coding technique, the image is partitioned into a number of non-overlapping range blocks. The larger domain blocks are selected from the same image and can overlap. A grayscale image can be coded by mapping the domain block D to the range block R with the contractive affine transformation [7] given in the equation 1.

$$R=i\{\alpha(So\,\Delta)+D\ g\} \qquad (1)$$

Where,

R - Coded range block

α - the scaling factor

S - the contractive transformation

D - Domain block

Δg - luminance shift

i - isometry

Then the fractal code describing the above contractive affine transform that has minimum matching error between the original range block and coded range block R are transmitted and stored .The fractal code consists of contrast scaling α, the luminance shift Δg, isometry i, position PD of the best matching domain block in the domain pool. In the decoding stage, an initial arbitrary image is selected and the decoded image is repeatedly reconstructed by applying the same contractive affine transformation to the iterated image.

Here the aim of the paper is to reduce the time complexity in the encoding process. This has been achieved using edge detection before the encoding phase. In addition mean image is transmitted along the fractal codes. So the transformations can be done at the decoding stage without any iterations leading to reduction in compression time.

2. EDGE DETECTION

An edge is a set of connected pixels that lie on the boundary between two regions [6]. The magnitude of the first derivative of the pixel intensity values can be used to detect the presence of an edge. To be classified as a meaningful edge point, the transition in pixel intensity values associated with that point has to be significantly stronger than the background at that point. Since it is local computation, the method of choice to determine whether a value is "significant" or not is to use a threshold. Thus we define the point in the image as an edge point if its two dimensional first order derivative is greater than the specified threshold. A set of such points that are connected according to a predefined criterion of connectedness is by definition an edge.

2.1. Gradient Operators

First order derivatives of a digital image are based on various approximations of the two dimensional gradient [6]. Gradient of an image f(x,y) at location (x, y) is defined as the vector given by equation (2).

$$\nabla f=\begin{bmatrix} G_x \\ G_y \end{bmatrix}=\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \qquad (2)$$

The gradient operator points in the direction of maximum rate of change of edge at coordinates (x,y). An important quantity in edge detection is the magnitude of this vector denoted by equation (3).

$$f = \text{mag}(f) = [G^2_x + G^2_y]1/2 \qquad (3)$$

This quantity gives the maximum rate of increase of f(x,y) per unit distance in the direction of f. This is referred to as the gradient.

The direction of the gradient vector $\alpha(x,y)$ at (x,y) is given by equation (4).

$$\alpha(x,y) = \tan^{-1}(G_y/G_x) \qquad (4)$$

The direction of an edge at (x,y) is perpendicular to the direction of the gradient vector at that point.

The computation of the gradient of an image is based on obtaining the partial derivatives at every pixel locations.

2.2. Laplacian Operator

The Laplacian of a 2 – D function f(x,y) is a second – order derivative defined as in equation (5).

$$\nabla^2 f = \partial^2 f / \partial x^2 + \partial^2 f / \partial y^2 \qquad (5)$$

Digital approximations to the laplacian for a 3 x 3 region is given by equation (6).

$$\nabla^2 f = 8 Z_5 - (Z_1 + Z_2 + Z_3 + Z_4 + Z_5 + Z_6 + Z_7 + Z_8 + Z_9) \qquad (6)$$

3. COMPRESSION AND DECOMPRESSION

3.1. Compression

To encode the image, it is partitioned into blocks. An image partitioning scheme is selected to generate range blocks Ri. The original 256*256 image is reduced by averaging into 64 * 64 image. This new image is called the Domain image. Both images are partitioned into 4*4 blocks. 4 * 4 blocks that are chosen as range blocks will result in higher compression time and those chosen as larger blocks will deteriorate the image quality.

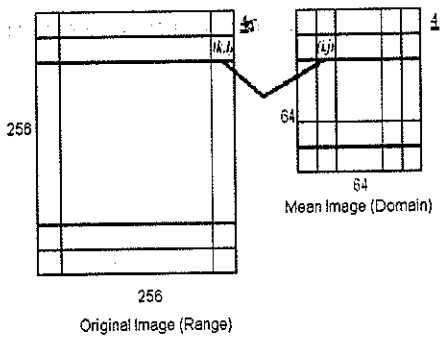


Figure 1: Obtaining mean image from original image

All range blocks are classified into two classes, either as a shade block or rough block according to the variance of the block. The variance of each range block can be calculated using the equation .

$$\text{Var}(R) = 1/B^2 \Sigma(r_{i,j} - \mu_R)^2 \qquad (7)$$

Where

R - range block
B - size of the block
$r_{i,j}$ - pixel intensity value at position (i,j)
μ_R - mean of the block

If the variance of the range block is less than a threshold value E_{th}, then the range block is of type shade block. The shade block can be approximated by a value equal to the average pixel intensity value of the block. These shade blocks are coded simply by the mean value of the range block. All other blocks of variance greater than the threshold value are termed as rough type blocks. The rough blocks are coded by affine transformations.

For each range block, a best matching domain block is searched. The affine transformation of the pixel values (that is, a scaling and rotation) is found that minimizes the MSE difference between the transformed domain pixel values and the range pixel values. The transformation can be represented by the equation.

$$g(x_1, x_2) = \begin{pmatrix} a1 & 0 \\ 0 & a2 \end{pmatrix} O \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

where

gp is affine mapping.

a_1 , a_2 are positive integers;

O is orthogonal transform that consists of a rotation of a multiple of 90 degrees .

(b_1, b_2) is the translational vector.

Given an image f to be encoded, a collection of transformations are found. The set of all transformations are stored.

For a fixed image, more transformations lead to better fidelity but worse compression ratio.

Finally , a header is attached for each range block to denote its coding status as either coded by mean or affine transformations. Therefore, the decoder can correctly reconstruct each coded range block according to the header. The fractal codes are transmitted and stored.

In fractal compression with edge detection [1], one of the edge detection masks is applied over the range and domain images before encoding is performed. The images with its edges detected are obtained in this manner. Each pixel in the image will be either part of an edge or an interior pixel. The range and domain blocks are now classified by checking whether any of the pixels in each block contains an edge pixel. In this way each block can be either an edge block or an interior block. While comparing between the range and domain block four cases arise:

If $\mathbf{r}_i$ is a range block and $\mathbf{d}_i$ a domain block

1. $\mathbf{r}_i$ and $\mathbf{d}_i$ are both interior blocks (*int/int*),
2. $\mathbf{r}_i$ and $\mathbf{d}_i$ are both boundary blocks (*bnd/bnd*),
3. $\mathbf{r}_i$ is an interior block whereas $\mathbf{d}_i$ is a boundary block(*int/bnd*),
4. **ri** is a boundary block whereas $\mathbf{d}_i$ is an interior block(*bnd/int*).

In the first case the range block does not contain an edge pixel. Hence it is compared with all blocks in the domain which has no edges. The mean squared error is calculated in each case and the minimum is chosen.

In the second case, the range block is an edge block, ie, it contains a pixel which is part of an edge. Therefore while coding only the domain blocks with edges are considered. In a similar way the mean squared error is calculated and the block with minimum error is stored.

The third and fourth cases which are also considered in the conventional fractal coding procedure, can be assumed to contribute little to the PSNR performance of the compression. It has therefore been eliminated in this paper. This reduces the computational complexity and the time taken for encoding is also drastically reduced.

3.2. Proposed algorithm for encoding

A proposed algorithm on how the paper is being implemented is given below:

flg and flge are flags used to indicate whether a block is an edge block or an interior block, in the original and mean images respectively.

1. Read the input image.
2. Obtain the mean image by averaging the original image.
3. Perform edge detection on the original image.
4. Perform edge detection on the mean image.
5. Partition the range image into fixed square blocks of size 4x4.
6. Partition the domain image into fixed square blocks of size 4x4.
7. Traverse through the range blocks,

   if any of the pixels in the block is an edge pixel,

   then

   set flg=1

   else

   set flg=0

8. Traverse through the mean image block, if any of the pixels in the block is an edge pixel,
   then
     set flge=1
   else
     set flge=0.
9. Calculate variance for each range block.
   If variance > threshold
   then
     go to step 10
   else
   store current block as shade block.

10. For the current range block,
     if flg=1
       perform comparison with each domain block having flge=1
     else
       perform comparison with each domain block having flge=0.
11. Perform various affine transformations.
12. Choose domain with minimum MSE.
13. Store the corresponding transformations.
14. Go-to step 7 until entire image is coded.
15. Stop.

3.3. Decompression

Decompression of an image is simple. The mean image and the fractal codes are stored in the fractal code book. The mean image is obtained from the code book. The stored transformations are applied to the mean image which is the domain image. Once all the transformations are performed, the decompressed image is obtained.

4. IMPLEMENTATION AND RESULTS

Fractal image compression with edge detection for both color and gray scale images in the spatial domain has been implemented in this paper. The range and domain block considered is of size 4 x 4. The image partition scheme considered is the fixed square partition. Affine transformation and scaling are used to compare the blocks. The Laplacian mask has been used for the edge detection. The image with the edge detected is stored in a separate array. As mentioned in the general algorithm, the flags flg and flge are used to indicate whether a block is an edge block or an interior block.

The images used for testing the program are of size 256 x 256, 24 bit bitmap color images and 8 bit bitmap gray-level images. The mean image size got from averaging the original image is of the size 64 x 64.

The performance is compared taking the following parameters into account:

i. Compression time
ii. Decompression time
iii. Compression ratio
iv. PSNR (Peak Signal to Noise Ratio)

4.1. Results

The table 1 shows the results for compression with edge detection and it is compared with the results obtained for compression without edge detection.

Image	Compression Ratio	Compression Time (Secs)	Decompression Time (secs)	PSNR (dB)
Dune	91.4447	27.8440	0.1710	36.0896
Lena	90.0408	61.9530	0.1880	28.9400
Parrots	90.5777	56.3750	0.1880	27.5618
Pepper	89.4392	53.4060	0.1870	26.0626

Table 1 Results for Compression with Edge Detection

Image	Compression Ratio	Compression Time (Secs)	Decompression Time (secs)	PSNR (dB)
Dune	91.4447	90.094	0.1880	37.3389
Lena	90.0408	151.188	0.1880	29.6798
Parrots	90.5777	127.588	0.1870	28.1308
Pepper	89.4392	176.063	0.1870	26.9581

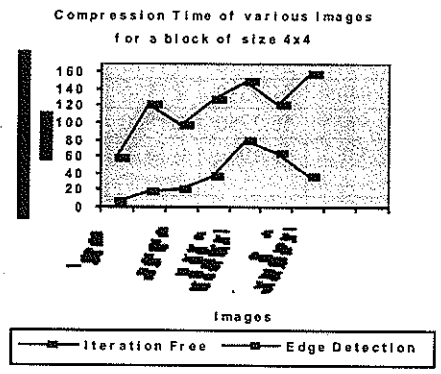
Table 2 Results for Compression without Edge Detection with Color Image

The comparison indicates that proposed compression method leads to significant reduction in the compression time and it also gives decompressed image with the good quality. The figure 2 (a) shows the original image, figure 2 (b) shows the decompressed image without edge detection and figure 2 (c) shows the decompressed image with edge detection. Figure 2(d) gives a comparison of the compression times for gray level Images

(a)

(b)

(c)



(d)

(a) Original image

(b) Decompressed Image (without Edge detection)

(c) Decompressed Image (with Edge detection)

(d) Comparison of Compression Times for Gray Level Images

Figure 2

References

[1] Kamel Belloulata and Janusz Konrad, April 2002, "*Fractal Image Compression with Region Based Functionality*," IEEE Transactions on image processing, vol. II, No. 4, pp. 1-12.

[2] Hannes Hartenstein, Mathias Ruhl, and Dietmar Saupe, July 2000, "*Region-Based Fractal Image Compression*," IEEE Transactions on Image Processing, vol. 9, No. 7, pp. 1171-1184.

[3] Arnaud E. Jacquin, January 1992, "*Image Coding on a Fractal Theory of Iterated Contracted Image Transformations*," IEEE Transactions on Image Processing, vol. 1, No. 1, pp. 18-30.

[4] B. Wohlberg and G. de. Jager, December 1999, "*A review of the fractal coding literature*," IEEE Transactions on Image Processing, vol. 8 No. 1, pp. 1716-1729.

[5] A. K. Jain, "*Image data compression: A review*" Proc. IEEE, vol 69, March 1981.

[6] Rafael C.Gonzalez and Richard E. Woods, 2003, "*Digital Image Processing*", 2nd edition., Pearson Education, Inc.

[7]. Hsuan T.Chang and Chung J. Kuo "*Iteration – free fractal image coding based on efficient domain pool design*," IEEE Transactions on Image Processing, vol.9 , No.3 , pp. 329-339, March 2000.

[8]. Raouf Hamzaoui and Dietary Saupe, "*Combining Fractal Image Compression and Vector Quantization*," IEEE Transactions on Image Processing, vol.9, No.2, pp. 197 – 208, Feb. 2000.

[9]. Hsuan T. Chang, "*Gradient Match and Side Match Fractal Vector Quantizers for Images*," IEEE Transactions on Image Processing, vol.11, No.1, pp.1-9, Jan. 2002.

[10]. Y. Linde, A. Buzo and r. Gray, "*Algorithm for vector quantization design*," IEEE Transactions on Communication, Vol.28, Jan 1980, P84-P95.