Adaptive Partial CDN Replication

Manos Spanoudakis
Dept. of Informatics & Telecommunications, University of Athens
e.spanoudakis@di.uoa.gr

Stathes P. Hadjiefthymiades
Dept. of Informatics & Telecommunications, University of Athens
shadj@di.uoa.gr

Abstract

Content Distribution Networks are a very important part of today's Internet and the Web. They enable the scalable provision of content and yield significant saving for the information provides (the clients of the CDN). In this paper we propose a scheme that tries to optimize the use of disk space in the CDN. The basic idea involves the partial replication of the contents of a web site. Site objects are transferred to the CDN in bundles. The site partitioning technique that produces bundles can be based on the site structure or the popularity of the site contents. We propose two schemes for site partitioning: the static partitioning and a technique based on Markov graphs (mostly intended for Web prefetching). We simulated the discussed schemes using extensive traces taken from the site of our department. Our findings are very promising for the proposed schemes.

1. Introduction

Content distribution networks (CDN) are becoming more and more important nowadays due to the impressive growth of the WWW and associated technologies. The need to efficiently push information close to the interested clients through a ubiquitous infrastructure is dictated by the ever increasing supply of information, the introduction of new Internet sites and the need of high specialization in today's corporate environments.

A CDN supports content providers on the highly important issue of scalable content delivery and delivers their content to any interested client [3],[4]. CDN adoption yields two important benefits to content providers. As discussed above, CDN presence is global (ubiquitous). Typically, CDN companies deploy nodes around the globe with computing and networking capabilities that are relatively difficult to saturate. The collaboration between content providers and content distributors allows the former to secure global presence with fairly reduced costs. Moreover, the CDN adoption protects the infrastructure of the content provider from unforeseen situations like the sudden peaks in the experienced load (flash events).

The CDN commits different types of resources to the content provision task. Resources like disk space, computing capacity and network bandwidth are devoted to the CDN clients (i.e. the content providers). An increased clientele of the CDN would imply increased probability for the saturation of the aforementioned resources. In this paper we focus on disk capacity as we believe that this resource type can be easily exhausted in contrast to the other types. Web content is growing in exponential terms i.e the size of the average web page has more than tripled [6] (grew from 93.7K to over 312K). Furthermore, the wide adoption of new multimedia formats like High Definition video, is expected to add more load on disk utilization and the inclusion of the CDN in the information delivery chain implies that the induced load is shifted to the CDN infrastructure.

As discussed above, our proposed solution tries to mitigate the disk space saturation risk. We try to rationalize the commitment of disk space to the different CDN clients. We focus on the solution of partial content replication i.e., not all the content available on the content provider is pushed to the CDN. Only certain segments of the sites are made available through the CDN. Their selection is based on different characteristics of the site like the structure (dependencies between objects) or usage monitoring (popularity). We capitalize on such information to reduce the volume committed to a certain provider in the CDN and, thus, increase the CDN capacity. Our solution implies the risk of missing a requested object from the replicated set found at the CDN. In this scenario, the user's request is passed on to the original site and served there. Hence, the interested user incurs a limited increase in the average retrieval latency. We try to quantify the pros and cons stemming from our architecture through a series of simulations, using the various algorithms proposed.

This paper is structured as follows. Section 2 presents the proposed architecture. Section 3 is devoted to the site segmentation/partitioning algorithms adopted for the formulation of WWW bundles. Bundles are transferred to the CDN to cater for user requests. Section 4 presents our simulation findings. Finally, Section 5 closes the paper by presenting our conclusions.

2. System Architecture

The proposed system architecture extends the classic CDN architecture to facilitate the partial replication of the site to the surrogates. As already discussed the proposed system aims to replicate a part of the CDN at the surrogates. This partial replication is performed by partitioning the site graph into subgraphs and replicating some of these subgraphs at the surrogates. A specific algorithm, according to the desired criteria, determines the selection of the subgraphs that are replicated. More details on the adopted algorithms are provided below.

One of the key tasks of the system is to partition the site in an effective way so as to achieve an increased hit rate, while occupying as less space as possible. To achieve this, a number of functionalities must be implemented, such as the “partitioning” and the “replacement” of the fragments on the surrogates. Furthermore, a number of messages should be exchanged between the CDN Server and the surrogates (the actual replication servers). The system architecture is depicted in Figure 1.

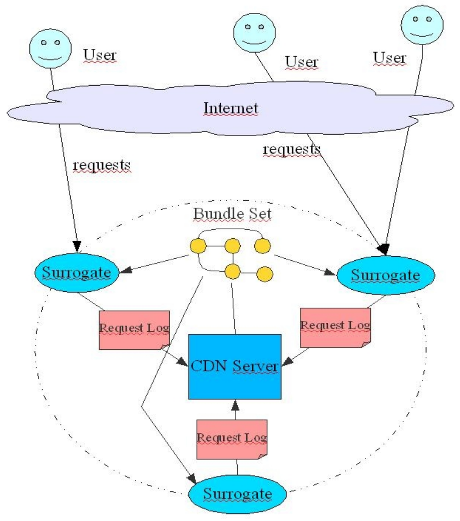

Figure 1: System Architecture

As depicted in Figure 1, the CDN server sends a message which contains the subgraphs (bundleSet) that should be replicated on each surrogate. It must be noted that a bundle is not a structure containing the actual objects, but a logical entity, i.e. a set of references to the objects it includes. The body of the message may only describe the actual objects to be replicated, or even contain these objects. In the first case, it is the surrogate's responsibility to retrieve these objects from the origin server. Such messages may be sent periodically, or only at system start-up, according to the partitioning algorithm that is used.

Another type of messages (request Logs in Figure 1) that is exchanged between the surrogates and the CDN server may be used, in some cases. Depending on the adopted algorithm data regarding the requests of users may be exploited in order to perform the partitioning. In this case, each surrogate sends a log file indicating the user requests that the surrogate was asked to serve. The CDN server then, executes the partitioning algorithm using the data from these logs in conjunction with previously processed data, re-partitions the site graph and sends a message to the surrogates containing the updated bundles set. As already mentioned, the surrogates may then need to perform a number of requests to the origin site to retrieve objects that are not already hosted.

As stated above, the proposed schema, aims to achieve a high hit ratio, while occupying as less disk space as possible on the surrogate. A request from a user is characterized as a hit when it the surrogate can serve it locally, i.e. the requested object is available within the bundles found in the surrogate. In case the surrogate fails to serve a request locally, it should contact the origin site and fetch the resource. In this case there is a small time overhead incurred by the user but this scenario occurs with very limited probability.

3. Site Partitioning Algorithms

The partitioning algorithms used to create the CDN bundles, can be distinguished in two categories according to the criteria used to perform the partitioning. The first set includes these algorithms that perform the partitioning based on the site structure, using metrics such as the interconnectivity of web pages, their “distance” (the number of hyperlinks the user needs to follow in order to go from one page to the other), etc. Such metrics are static, and therefore, not affected by any other parameter but the structure of the site.

On the other hand algorithms, belonging to the second set use dynamic data to derive metric values. Such a dynamic algorithm relies on analysis of web traces in order to determine the relativity (the possibility to navigate from one page to the other based on the user behavior within the site) of objects based on visitor statistics. In our implementation, we have included two algorithms one from each category:

"Static Site Analysis" (SSA) as a representative of the static algorithms category, and, "Markov Site Analysis", as a representative of the dynamic schemes.

The details of the aforementioned algorithms are discussed below.

3.1. Static Site Analysis

The concept of the algorithm is to isolate "central" nodes (objects) of the web site and use each one of them as the base point of a bundle. Each node is used as a starting point to perform a breadth first walk in the graph. Every node visited is added to the specific bundle, until it is filled up, or there are no remaining nodes to visit. The main steps of the algorithm are shown in Figure 2 and discussed in the following paragraphs.

```
SiteGraphGenerator("www.di.uoa.gr")
SiteGraph=SiteGraphHandler.readGraphXML("XML
_GraphFile")
SortedVerticesList
=SiteGraph.sortVerticesDegree()
Bundles=SiteGraph.generateBundles(SortedVelt
icesList)
```

Figure 2: Main Steps of the Static Site Analysis

As shown above the algorithm is divided in four main steps. The first step (method SiteGraphGenerator()), utilizes a web spider object to analyze the target web site and produce an XML representation of the site graph. The web spider used was based on the WebSPHINX library [1], which was adopted to meet the specific requirements of the test case. In more details, the aforementioned method, visits every resource (URL) of the site, and produces an XML file representing the structure of the site as a directed graph i.e. a set of vertices and a set of directed edges. A part of the resulting XML file is shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SiteMap SiteUrl="http://www.di.uoa.gr">
  <VerticesList>
<Vertex url="http://www.di.uoa.gr" size="2260" />
<Vertex  url=http://www.di.uoa.gr/images/menu-uoa.gif
size="9488" />
<Vertex url="http://www.di.uoa.gr/gr/" size="5562" />
  . . .
  </VerticesList>

  <EdgesList>
<DirectedEdge source="http://www.di.uoa.gr"
target="http://www.di.uoa.gr/images/menu-uoa.gif" />
<DirectedEdge source="http://www.di.uoa.gr"
target="http://www.di.uoa.gr/index-items/di-
grlft.jpg" />
<DirectedEdge source="http://www.di.uoa.gr"
target="http://www.di.uoa.gr/gr/" />
        <DirectedEdge source="http://www.di.uoa.gr"
target="http://www.di.uoa.gr/index-items/choffgr.jpg"
/>
  . . .
  </EdgesList>
</SiteMap>
```

Figure 3: XML representation of site structure

The next step is to read the resulting XML file and generate a graph object, which will be used to create the bundles. This functionality is implemented in the "SiteGraphHandler.readGraphXML()" method.

The produced graph object needs to be further processed before the final generation of bundles. As already mentioned the algorithm isolates the most "central" nodes of the graph and generates clusters-subgraphs, starting with these ones. The centrality metric is calculated in this step, as the number of outgoing links from each graph node. Furthermore, all nodes are sorted based on their centrality, in descending order. This functionality is implemented in the method "SiteGraph.sortVerticesDegree()" which is presented in more details in Figure 4.

```
sortVerticesDegree()
{
/* Sorts the Vertices of a graph based on the
number of outgoing links from a node in descending
order. */

For each Node
    Degree=Node.outDegree
    NodesArray.add(Node,degree)
End for
NodesArray.sortDescending
}

bundlesArray[] generateBundles()
{
  i=0
  while i< NodesArray.size()
  {
    rootNode=NodesArray(i)
    currentBundleSize=0
    bundle = new Bundle
    nextNode=rootNode.breadthFirstIterator.Next()
    while ((nextNode != Null)
           and
          (nextNode.size+ currentBundleSize <
                                 maxBundleSize )
         )
    {
      bundle.add(nextNode)
      currentBundleSize += nextNode.size
    }
  }
}
```

Figure 4: Bundle generation process

Finally, the last step of the algorithm is to generate the bundles resulting from the Static Site Algorithm. The sorted graph nodes processed in the previous step are used to generate the bundles. Starting with the most "central" ones each node is used as a root node to perform a breadth first walk in the graph. All nodes visited during a walk are added to the same bundle until it is filled up or there are no more nodes to visit. The method terminates, when a specified number of bundles (passed as a parameter) are filled in. Method "generateBundles" (pseudocode also in Figure 4) performs the appropriate processing in the sorted nodes produced in the previous step, and generates an Array object containing the outgoing bundles (i.e., the bundles to be sent to the surrogate).

It must be noted that due to the structure of a web site graph, there are circles, and therefore a breadth first iteration starting from a different node each time, may visit nodes that have been visited in previous iterations (that have been initiated with different root node). The implemented algorithm takes special care to avoid the duplication of nodes in distinct bundles.

3.2. Markov Traffic Analysis

In contrast to the previous algorithm, which is based on the static structure of a web site, the second algorithm we examine is a dynamic one. In more details, the Markov Traffic Analysis algorithm generates a Markov graph based on past user requests. It is characterized as a dynamic algorithm, since the graph used for partitioning of the web site is regularly updated with new users' request data at runtime. Therefore this algorithm is adaptive to live user interaction, allowing users' "trends" to affect the partitioning scheme and the respective generation of bundles.

The implemented algorithm utilizes a Markov dependency graph, which is constructed as described by Padmanabhan and Mogul in [2]. The weighted graph is constructed in order to depict users' access patterns, and it is used to partition the designated site into clusters-bundles that have "proven" to be widely visited paths. In other words, the algorithm attempts to isolate nodes that are regularly visited "sequentially" by users. The dependency graph is updated as new requests arrive, and in regular intervals the site is repartitioned in bundles that reflect the latest user trends in movements within this site.

As shown in Figure 5, the partitioning algorithm is executed in two phases. It must be noted that the two phases can be executed asynchronously, meaning that phase 1 may continue to execute in parallel with phase 2, and does not have to halt until the later completes. In Phase 1 (MarkovGraphHanlder.addRequest()), all the logged requests are stored in an array.

```
While true(){
  MarkovGraphHanlder.updateGraph()
  If (interval) then{
   SortedVertListMarkovGraphHanlder.
                  sortVerticesWeight()
   Bundles= MarkovGraphGenerator.
                  generateBundles(SortedVertList)
   }
}
```

Figure 5: Partitioning Algorithm

In regular intervals (algorithm parameter), the second phase of the algorithm is triggered. This phase involves the update of the Markov graph, with new user requests' data, the sorting of the graph based on a specific metric and finally the bundle generation. In more details, the update of the Markov graph (method "MarkovGraphHanlder.updateGraph()") is performed as follows. All data regarding user requests saved in the array in phase1, are used to identify all vertices of the graph that are "affected" i.e. are included in the array. For every vertex in the array, all outgoing edges' weights are updated accordingly. Subsequently (method "MarkovGraphHanlder.sortVerticesWeight()" in Figure 6) the graph is sorted on descending order based on a metric, which is assigned to every node of the graph. This metric, labeled D, is calculated as follows:

$$D=\sum_{1}^{N} edge.weight$$

where *edge.weight* in the above equation is the weight of each outgoing edge, as calculated by the aforementioned method "updateGraph()". Finally method "MarkovGraphGenerator.generateBundles(SortedVert List)" is invoked to generate the bundles that will be used to replicate the site. The bundles are then replaced accordingly to the surrogates.

```
While true(){
    MarkovGraphHanlder.addRequest()
    If (requests=interval) then {
      MarkovGraphHanlder.updateGraph()
      SortedVertList= MarkovGraphHanlder.
                         sortVerticesWeight()
      Bundles= MarkovGraphGenerator.
                 generateBundles(SortedVertList)
      Requests=0
    }
    requests = requests + 1
}
```

Figure 6: Monitoring and Graph Generation algorithms

It must be noted that this algorithm requires a warm-up period, during which all incoming requests are handled as "misses" to the "request analyzer" object in order to initialize the graph and perform the initial partitioning of the site. It must be noted that during this period there is no replication of the site at the surrogates. However, this can be avoided if previous request data collected prior to system installation are collected and analyzed, and, thus, used to perform the initial partitioning of the system.

4. Simulation Results

In order to assess the performance of the proposed schemes we have undertaken a series of simulations. Our simulations involved the replication of an extensive web site (the site of our department). We assumed that the site content was partitioned according to the algorithms presented in previous sections and monitored the behavior of the replication scheme. We have tried to benchmark the proposed algorithms through an extensive, real web access log taken from the same site (captured in 3 months period). We have evaluated different levels of partial replication (percentage ranging from 48% to 72%). The bundle size was either 5 or 10 MB. The total volume of the web site was 250 MB.

We monitored the performance of the CDN w.r.t. the object hit rate (OH) achieved in light of the partial replication. Apart from the object hit rate, we also monitored the byte hit rate (BH), another popular metric for caching systems. We use the metrics OG (Object-level Gain) and BG (Byte-level Gain) defined as follows:

$$OG = \frac{OH}{rp} \qquad BG = \frac{BH}{rp}$$

In the above metrics, rp denotes the replication level. Higher gain values (OG/BG) are desirable indicating that an increase in the hit rates and decrease in the rp. Evidently this means that the proposed scheme has achieved better hit rates using less disk space. Hence, when any of the above metrics increases so does the efficiency of the considered schemes . From Figure 7 and Figure 8 we can deduce that both schemes accept quite similar results in terms of the OG. The static partitioning scheme outperforms the Markov solution w.r.t. the BG. In Figure 9 and Figure 10 we plot the hit rate (object/byte) accomplished by the two schemes. We observe that the Markov partitioning performs similarly to the Static partitioning w.r.t. the OH. In all schemes, the OH approaches the 90%. In the Markov partitioning scheme, the BG remains almost stable at 50%. BG is considerably higher in the Static Partitioning case, indicating a preference of the algorithm to smaller sizes which are easily accommodated in the produced bundles.

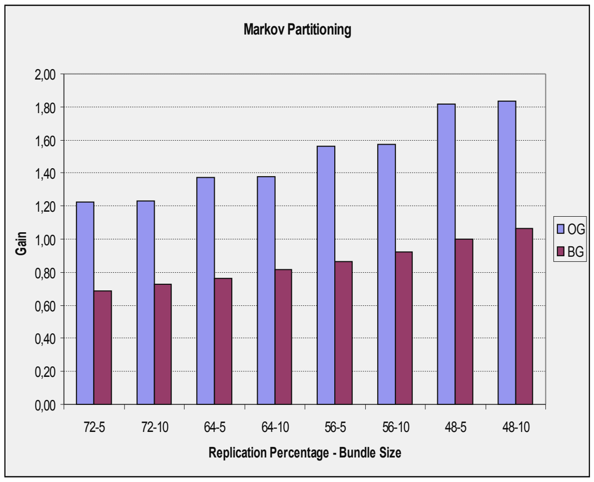

Figure 7: Object/Byte Gain for the Markov Partitioning

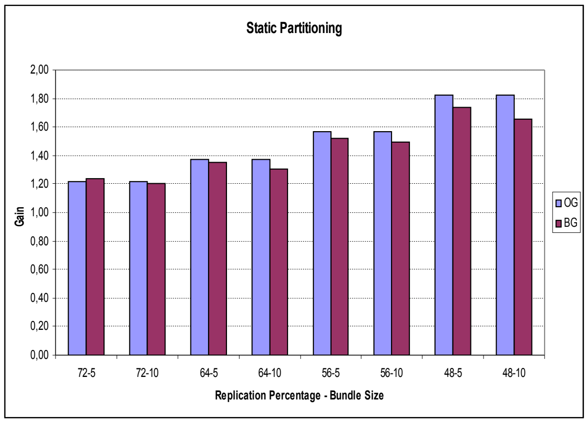

Figure 8: Object/Byte Gain for Static Partitioning

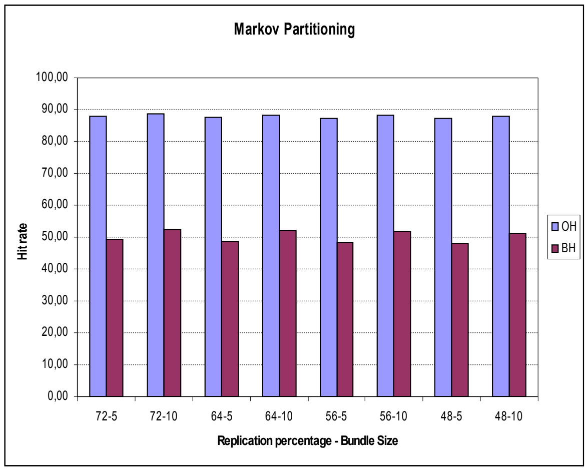

Figure 9: Object/Byte Hit Rate for Markov Partitioning

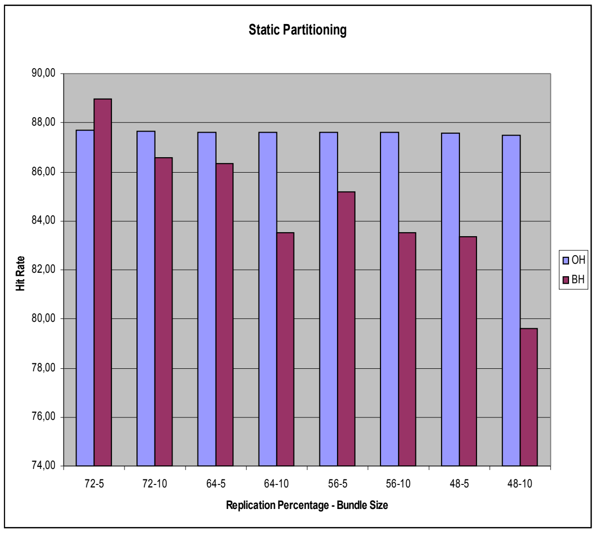

Figure 10: Object/Byte Hit Rate for Static Partitioning

5. Conclusions

In this paper we studied the partial content replication in CDNs. We presented an architecture which aims at optimizing the disk space while trying to minimize the miss ratio. In our implementation two algorithms were adopted, with different characteristics. The static algorithm is based on the structure of the web site, and the dynamic algorithm which tries to adapt to the user behavior experienced within the web site. Following the implementation of the algorithms we performed a series of simulations with extensive access traces.

Our results were very promising for all the involved parties. Currently we are working on more algorithms, one based on the structure of the web site and an adaptive one.

6. References

[1] A Personal, Customizable Web Crawler http://www.cs.cmu.edu/~rcm/websphinx/
[2] V. Padmanabhan, and J. Mogul, "Using predictive prefetching to improve World-Wide Web latency". In Proceedings of the ACM SIGCOMM Conference, 1996.
[3] M. Rabinovich and O. Spatscheck, "Web Caching and Replication", Addison Wesley, 2001.
[4] Markus Hofmann and Leland Beaumont, "Content Networking Architecture, Protocols, and Practice" Elsevier, 2005.
[5] Scot Hull, "Content Delivery Networks: Web Switching for Security, Availability, and Speed", McGraw Hill, 2002.
[6] Andrew King, "Website Optimization", O'Reilly Media, 2008