WebGraph: A Framework for Managing and Improving Performance of Dynamic Web Content[*]

Prasant Mohapatra and Huamin Chen
Department of Computer Science
University of California
Davis, CA 95616
{prasant, chenhua}@cs.ucdavis.edu

January 10, 2002

Abstract

The proportion of dynamic objects has been growing at a fast rate in the world wide web. In the e-commerce environment these objects form the core of all web transactions. However, because of additional resource requirements and the changing nature of these objects, the performance of accessing dynamic web contents has been observed to be poor in the current generation web services. We propose a framework called *WebGraph* that helps in improving the response time for accessing dynamic objects. The WebGraph framework manages a graph for each of the web pages. The nodes of the graph represent *weblets*, which are components of the web pages that either stay static or change simultaneously. The edges of the graph define the inclusiveness of the weblets. Both the nodes and the edges have attributes that are used in managing the web pages. Instead of recomputing and recreating the entire page, the node and edge attributes are used to update a subset of the weblets are are then integrated to form the entire page. In addition to the performance benefits in terms of lower response time, the WebGraph framework facilitates web caching, QoS support, load balancing, overload control, personalized services, and security for both dynamic as well as static web pages. A detailed implementation methodology for the proposed framework is also described. We have implemented the WebGraph framework in an experimental set-up and have measured the performance improvement in terms of server response time, throughput, and connection rate. The results demonstrate the feasibility and validates a subset of the advantages of the proposed framework.

Keywords: Dynamic Content Caching, Internet, Quality of Service, World Wide Web, WebGraph, Weblet, Web Servers.

[*]This work was supported in part by the National Science Foundation under Grant No. CCR-9988179.

1 Introduction

Performance and management of web servers has been a very active area of research in recent years. Several techniques have been proposed for improving the performance and management of web services. Some of the most common techniques that have been proposed include mirroring, caching web contents at proxy servers, distributed server farms with load balancers. These approaches are effective for web sites that have predominantly static web contents. However, all of these techniques are limited in terms of handling dynamic requests, scalability, overload, personalized services, and Quality of Service (QoS) assurances.

With the increase in the web usage and applications, several challenges are being faced in the web environment. The proportion of dynamic web contents have been increasing in most web sites. E-commerce has been a major business model in the expanding economy. In these environment, almost all of the web pages are dynamic in nature. Dynamic contents change with respect to time and events in varying granularity or with respect to the nature of queries. Processing of dynamic requests is usually compute-intensive and could be network-intensive if it needs accesses to back-end servers like databases, application servers, etc.. Because of the changing nature of dynamic web objects, they are usually not cached and thus do not exploit the caching advantages of proxy servers. Since most dynamic requests need the access of back-end servers, mirroring and load distribution techniques does little good for these type of requests.

The current generation web service used in the e-commerce environment suffers from several serious problems. Because of the presence of a high proportion of dynamic contents, the response time for accessing these sites has been very poor [6]. Both network and server contribute to the response delay. Poor response delay lead to significant revenue losses in e-commerce environments. The revenue loss in 1998 was estimated to be 1.9 billion dollars owing to long response delays [38]. In addition, overload conditions have serious impositions on the performance of web servers. Overload situations can degrade the server performance drastically, cause denial of services, and in some cases, crash the server. These situations arise because of the inherent nature of Internet traffic that includes unpredictability, burstiness, short-term cyclic shifts (e.g. hourly, daily, and seasonal variations, etc.). Applications with higher variability in resource requirements and lower delay tolerance are dominant contributors to the overload situations. Web environment with more of dynamic and multimedia components certainly add to the server load. Furthermore, QoS support through service differentiation and personalized services are highly desirable features in most web sites, especially the ones used in commercial environments.

An examination of several dynamic pages on the web revealed that in most cases, only a part or a few parts of the pages are dynamic in nature. Other portions of these pages constitute static images or text. However, for every access of the dynamic pages, the entire page gets constructed and rendered to the clients or the proxies. Thus the static characteristics of these pages are not being exploited in the current model of web access. Our initial motivation was to exploit this nature of the dynamic pages. Thus we developed a framework, called *WebGraph* that uses a graphical representation of web pages to serve dynamic pages very efficiently. Parts of the web page that the same attributes are called *weblets*. The weblets can be static or dynamic and are used to construct and reconstruct the web pages. In addition, we have also enriched the framework such that it can be used for other important attributes such as overload control, QoS assurances, caching efficiency, personalized services, and load balancing in web servers. This enrichment is achieved by attaching attributes to the graph elements (nodes and edges) in the framework. WebGraph interacts between the primary server(s) and the proxies or edge servers while facilitating the service of requests from the clients.

In addition to the performance benefits, overload control, and QoS support, WebGraph also facilitates personalized and value-added services, easy management, and publishing of web contents. The paper discusses and justifies these claims. We also outline the implementation and management details of the framework in addition to the discussions on its impact on the performance, caching behavior, overloads, and QoS. The WebGraph framework is compatible with the current generation web structure and can be deployed transparently in an incremental manner.

We have implemented the WebGraph framework in an experimental set-up and have examined its performance impact on workloads generated using the WebStone benchmark. It is observed that the response time, throughput, and connection rate of a WebGraph-based server is significantly lower than those of the traditional PHP-based servers. Furthermore, the experimental study validates the feasibility of the implementation of the proposed framework.

Although throughout this paper, we have portrayed WebGraph to aid the services of dynamic web content, it certainly provide all of the same advantages to the sites that have only static contents. Furthermore, WebGraph is not necessarily a competing framework; it can be used in conjunction with some of the other frameworks discussed in the related works (Section 6). However, to the best of our knowledge, there has been no framework reported thus far with as many features as provided by WebGraph. In that respect, it is the first of its kind.

The rest of the paper is organized as follows. The WebGraph framework is discussed in detail in Section 2. The usage, impact, and benefits of the scheme is discussed in Section 3. Section 4 outlines the implementation and management issues. The experimental implementation and performance measurements are discussed in Section 5. The related work is discussed in Section 6, followed by the concluding remarks in Section 7.

2 WebGraph Framework

The primary motivation of the WebGraph framework is to reduce the response delay for dynamic requests and lower the service demands on the Internet servers. The framework will enable faster access, efficient caching, lower bandwidth utilization, low I/O demands, QoS provisioning, overload control, personalized services, and ease of management of web services.

The main idea of the WebGraph framework is to divide the dynamic web pages into multiple components known as weblets. Web pages can be formed from these weblets by creating templates in a markup language. Weblets typically represent parts of a dynamic page where changes do not happen (static parts) or happen concurrently (dynamic parts). Thus, a weblet can be a static object (representing an image or text) or a dynamic object (representing a part of the page where data changes dynamically). Upon access to a dynamic page, instead of recreation of the entire page, only the weblets (based on their attributes as discussed later) that have changed are recreated and integrated with the precomposed partial page.

Unlike other scripting languages like Active Server Pages [2] and JavaServer Pages [30] in which all the scripts within a dynamic page must be executed serially, in WebGraph these components are capable of being executed independently in parallel and apart from the template. Thus any change in a dynamic page is reflected by re-executing only the related weblet(s).

A web page can have multiple weblets, and conversely a weblet can be embedded in multiple web pages. Furthermore, a weblet can include one or more weblets. A web page can be represented as a directed graph where the nodes are weblets and the edges represent the inclusion relationship. A directed edge from weblet-A to weblet-B indicates that weblet-B is included in weblet-A. Several attributes are defined for each of the nodes as well as the edges. Node attributes are different from the edge attributes. Node attributes refer to the properties of the weblets, whereas the edge attributes define the control information regarding the inclusion of the node in rendering the web page. These attributes provide information to the primary and proxy servers on how to manage the data efficiently. The attributes could include information regarding the caching nature, security, quality of service, time to live, and any other related information.

To clarify the concept and the structure of the WebGraph, let us consider an example of the web page of a hypothetical company xyz.com shown in Figure 1. The top and bottom of the pages include advertisements. The global advertisement (GA) targets a national audience, and the local advertisement (LA) targets the audience within a smaller region. Like the network television (i.e., ABC, NBC, CBS, etc.), which allows the local channels to include local advertisement, WebGraph facilitates the proxies to include localized advertisements or any other information (static or dynamic) that are of local interest. The site also contains news articles and weather information in addition to the display of transactions on which the company thrives. Each of these components contain static as well as dynamic elements. For example, the news would contain textual links, static images, and stock indexes, that need updating at different time intervals. Similarly the transactions will have components that will change with respect to certain events or actions.

In the WebGraph framework, the page shown in Figure 1 is represented as a graph as shown in Figure 2. The nodes represent different weblets and the edges denote their inclusiveness. The direction of the edges define the dependence relation of the weblets. The "News" weblet includes static images, textual links, stock indexes and the site logo. All of these weblets are static excepting the stock indexes. The weblet with site logo is included in multiple weblets. In the WebGraph of Figure 2, the dynamic weblets are stock indexes, radar images, current conditions and forecast, account profile, shopping cart, GA, and LA.

As mentioned earlier, both nodes and edges have attributes. The node attributes are marked as W1–W14, and the edge attributes are marked as L1–L15.

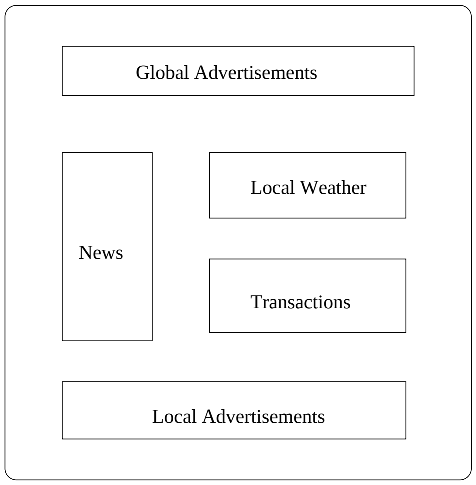

Figure 1: An example web page for a company xyz.com.

The possible attributes for nodes are tabulated in Table 1 (should be considered just as an example). We have shown four different attributes for the nodes: dependency, update period, QoS, and security. There could be more or less number of attributes depending on the applications and the web site composition. The dependency attribute defines the validity of a weblet, which means that if all the weblets on which it depends on (is composed of) are valid then it is considered valid. A weblet is considered valid if its TTL has not expired and its other attributes are satisfied. The leaf nodes have no dependencies. The second attribute is the updating period, which defines the length of time for which a specific data or object is considered valid. This attribute translates to the TTL factor in the implementation. A node is considered invalid if its TTL has expired, and will need to be retrieved from the primary server. An update period of "0" means that the object is not cached and is retrieved from the primary server for every access. The nodes that have dependent nodes do not have any update period as they are not updated as a whole. The third attribute for the nodes is the QoS-level. As an example, we have considered three levels of QoS: A, B, and C, corresponding to delay-sensitiveness, throughput-sensitiveness, and loss-sensitiveness, respectively. For example, the stock index node would be delay sensitive, the radar image node would be throughput sensitive, and the account profile would be loss sensitive. The nodes with dependencies would have the QoS field as blank because their QoS would depend on the QoS requirements of the dependent weblets. The fourth parameter in Table 1 refers to the security requirements of a site. Some weblets may require the use of a secure protocol (like SSL or TLS) and authentication while others may be accessed by anyone. The security attribute makes that differentiation. A node that has dependable nodes with different security level is marked as "mixed". We must emphasize here that we are citing these simple examples for the sake of explanation, and so they should be considered in the same spirit. QoS parameters with better matching of the application needs and other levels could certainly be conceived and supported by the framework.

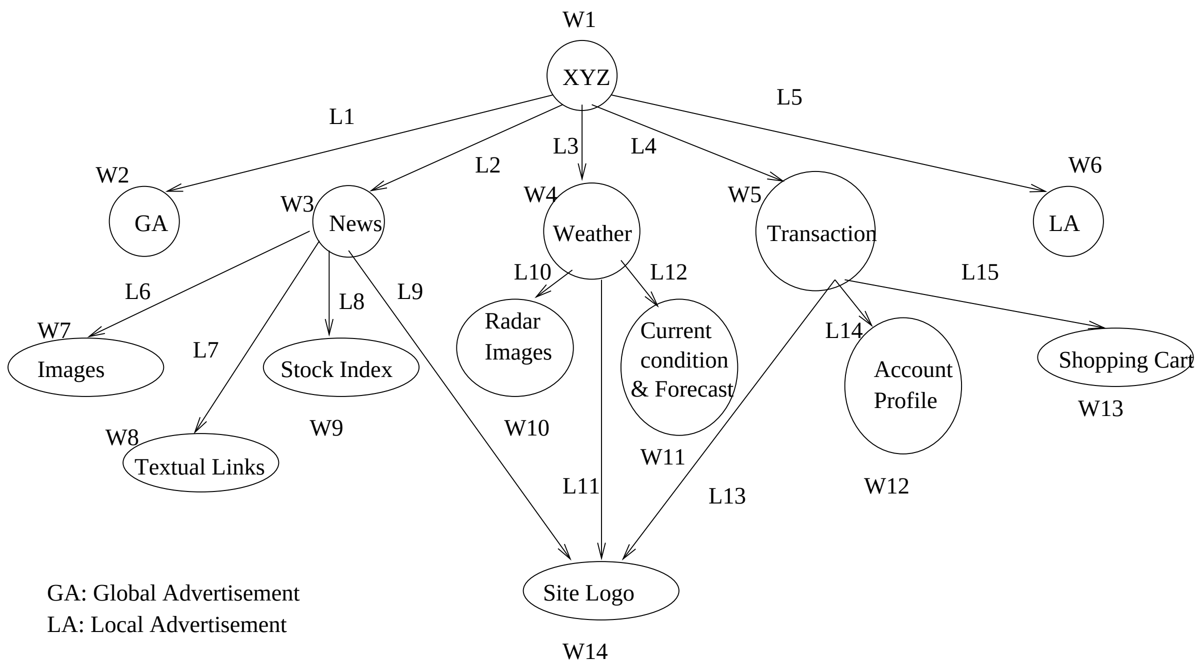

Figure 2: The WebGraph for the web page of xyz.com.

Table 2 shows the possible attributes of the edges. Although these attributes are shown in a binary mode in our example, it need not have to be that way. The attributes in all the three columns indicate whether an edge would be removed or not on the basis of an increase of load, inability to satisfy the required QoS, or due to security reasons. For example, under high server load, L6 and/or L10 could be removed. Similarly if the required QoS cannot be satisfied then some objects may provide incorrect or useless information. A delayed stock index value may not be useful or a very low quality radar image may be incomprehensible and thus be useless. The third attribute controls the security aspects. In case of any breach in security or while servicing in an insecure mode, some links should be removed. These four attributes may be changed and other attributes can be added based on the application environment. Our intent is to emphasize on the concept of the proposed framework and the examples used may not be the ideal ones.

The usage of the proposed framework is described as follows. WebGraph interacts between the primary server (the source server) and the proxies and/or the edge servers (e.g. Akamai servers [3]). For the ease of explanation, we do not distinguish between the proxies and the edge servers. The implementation and the support mechanism of WebGraph is the same for both of these types of servers. Thus unless otherwise specified, in this paper, the description of issues related to proxies are also applicable to the edge servers. WebGraph is usually created and updated at the primary server. The proxies maintain and can modify parts of the WebGraph as mentioned later. When a request from a client comes to the primary server through a proxy, the primary server sends the requested page and the corresponding WebGraph (the graph with all its attributes). The proxy stores the WebGraph and the weblets constituting the page, and while rendering the

Weblet	Dependency	Update-Period(min)	QoS	Security
W1	W2,W3,W4,W5,W6	-	-	mixed
W2	-	10	B	no
W3	W7,W8,W9,W14	-	-	no
W4	W10,W11,W14	-	-	no
W5	W12,W13,W14	-	-	mixed
W6	-	5	B	no
W7	-	1440	B	no
W8	-	60	C	no
W9	-	5	A	no
W10	-	15	B	no
W11	-	30	C	no
W12	-	0	C	yes
W13	-	0	C	yes
W14	-	10,000	B	no

Table 1: Node (weblet) attributes of the WebGraph.

Edges	Load-sensitive	QoS-sensitive	Security-sensitive
L1	yes/no	no	no
L2	yes	no	no
L3	yes	no	no
L4	no	no	yes
L5	yes/no	no	no
L6	yes	yes	no
L7	yes	no	no
L8	yes	no	no
L9	no	yes	no
L10	yes	no	no
L11	no	yes	no
L12	yes	no	no
L13	no	yes	no
L14	no	no	yes
L15	no	no	yes

Table 2: Edge attributes of the WebGraph.

page in response to future client requests, uses the WebGraph framework. The overhead for updating changes in the WebGraph is insignificant since the graph topology is unlikely to be changed frequently.

The proxies serve requests using the WebGraph, which reduces the bandwidth demand on the Internet, expedites the client response time, while reducing the load on the primary server. In the WebGraph framework, a considerable amount of load is transferred from the primary server on to the proxy servers. Thus server bottlenecks can be avoided using this framework, and thus the throughput of the primary server will increase.

The WebGraph framework has several advantages in managing and rendering a variety of web sites. Some of these advantages are discussed in the next section; several more can be perceived and supported through the framework.

3 Impact of WebGraph

In this section we describe the usage of WebGraph framework and how we can derive benefits from it in terms of performance, quality of service support, web caching, load balancing, overload control, personalized services and security.

3.1 Performance Impact

The primary performance parameter in web service is the response time. WebGraph is likely to reduce the response time of web requests significantly because of two reasons. First, the weblet-based approach allows for partial processing of web pages and reduces the unnecessary recomputations of the entire page comprising of one or more dynamic weblets. Thus the processing of complex page structures would be faster. The overheads associated with the implementation of the WebGraph framework is not high as discussed in Section 4. The second reason for the faster response time would be due to the transfer of load (to a certain degree) from the primary server to the proxies. This load transfer reduces the load on the primary server which is more likely to be the bottleneck compared to the proxies. The load transfer also reduces the bandwidth demand on the Internet as the requests and file transfers from the primary server (mostly over the Internet) is reduced. Only the specific weblets are recomputed and sent across (from the primary server to the proxies) rather than transferring the entire page all the time. Since the proxies are geographically closer to the clients, it makes more sense to transfer additional load on to the proxies, thereby reducing the bandwidth demand on the longer access path to the primary servers.

3.2 Impact on Web Caching

Although web caching is a very effective technique for improving web performance and has been a very active area of research for the past few years, there has been very little work reported on the caching issues of dynamic pages. In most contemporary web services, dynamic pages are not cached. Because of high volatility and application-specific nature of dynamic pages, the caching issue in dynamic object is more complicated than its static counterpart.

WebGraph provides an efficient solution to the problems in caching dynamic web pages. Since the proxy has the image of the graph that forms the page, all the weblets are cached at the proxies. Upon access to a page, the proxy servers checks the validity of the weblets by examining the node and edge attributes. Only the invalid weblets are requested from the primary server or back-end servers and are used in reconstructing the page, which is rendered to the clients. Since the weblet attributes define their validity, the capacity of the web cache can be utilized very efficiently. Thus, by using WebGraph, dynamic requests can be cached, and only parts of it are retrieved from the primary servers as and when necessary.

3.3 Quality of Service Support

The QoS support can be handled in a much efficient manner in the WebGraph framework. One of the attributes of the weblet could define the QoS parameter as described in our example in the previous section. The QoS could be related to delay, throughput, loss rate, or any other service parameters. Thus for each of the weblets, the QoS constraint can be imposed for better delivery of data. A weblet that has strict timing requirements is expected to be retrieved in a timely manner. A time-sensitive weblet (where the delayed values are of no use) may get dropped if delayed. Similarly for dynamic images (e.g., directions map from Mapquest.com) the QoS support can be provided through resource allocation using the WebGraph framework.

The edge attribute for QoS-sensitivity defines the links that can be removed if the required QoS requirements cannot be satisfied. For example, a poor quality radar image may be incomprehensible. Thus a significant amount of resources may be saved rather than getting used in delivery of data that is of no use. In other words, if the quality of an object falls below certain level of QoS because of resource unavailability, the edges from the WebGraph are removed and those objects are not fetched. This approach results in saving of resources, which could be used by other objects.

Another approach for QoS support can be also provided by WebGraph. Different versions of the weblets could be used for representing different quality of the data or image. Based on the QoS requirement, the appropriate weblet can be used for creating the web page. Thus depending on the QoS requirement, different sets of edges will be added or removed. In addition, attributes can also be used for service differentiation; preferred customers are given additional or different weblets representing value-enhanced services, etc.

It is being predicted that there will be an influx of handheld wireless devices surfing the Internet. This scenario will create a different problem. The handheld devices may be limited in resources (bandwidth, buffering capacity, power consumption, processing ability) and thus may not be able to handle the rich contents of a web page. Using the WebGraph framework, we can have the attributes of some of the weblets configured to match the needs and resource availability of the clients. For example, dense and high-quality images can be replaced by lightweight and lower quality graphics if they do not hinder the purpose. Thus for such applications, by using the edge attributes, we can reconfigure the topology (by removing weblets corresponding to dense contents and adding weblets corresponding to the lightweight contents) of the graph in the WebGraph framework and render the pages effectively and efficiently.

3.4 Load Balancing

Load balancing is a big issue for large-scale web servers and in proxies that provide scalable services for large enterprises. In these environments, typically a cluster of nodes is used with the goal of providing web services in a load-balanced manner. Several approaches have been proposed and deployed for balancing the load in these clustered environment. The WebGraph framework does not hinder any of those approaches; rather, it facilitates the implementation and management of load balancing objectives.

For traditional web applications, the whole process must reside on one server irrespective of their capacity or load. In the WebGraph framework, the weblets within a dynamic page can be distributed on different servers according to their capacity and workload. Some CPU intensive work can be run on servers with high frequency CPU's. Likewise, I/O intensive work can be run on those with high I/O bus capacity. In addition, the affinity of the weblets with respect to the nodes can be exploited to better utilize the caching effect at the nodes as proposed earlier in [36]. The scheduling is more flexible and robust than other load balancing schemes. The load balancing approaches using the weblets can be used for clusters both at the primary server as well as at the proxies. At the proxy server clusters the distribution of weblets may ease the manageability and distribute the network access load evenly. Weblets with very short update duration should be spread on different nodes, otherwise a single node may get choked with network accesses to reach the server for updates. Load balancing on the basis of other constraints, such as content-awareness [9], can be achieved through the proposed framework.

3.5 Overload Control

The throughput of an overloaded server could drop down to single-digit numbers and the server may even crash due to the overload. The response time from overloaded servers is usually unacceptable. As a result, a number of requests are aborted causing severe financial losses in the e-commerce environment. Not much has been reported on the overload control issues in web servers. In fact, most of the current generation e-commerce servers do not employ any overload control support. In the absence of this support, these servers are prone to cause denial of services. In the WebGraph framework, an attribute can be added at the link level, which gets used during the overload situations. Another approach is to have a second version of lightweight weblets for some of the objects that can be comprehensible at a coarser granularity. These attributes are labeled on the basis of the relative rendering priority of weblets during overloaded configurations. For efficient overload control, the web server must take action before the occurrence of the overload situation. Thus a load indicator can be used to indicate the onset (likelihood of overload in near future) of overload. With the onset of an overload, the primary server or the proxy server can use the attributes of the edges to decide what edges can be removed entirely or be replaced with another edge with a lower resource demand. Based on the load index, a prioritized approach of eliminating edges can be deployed. Many other perceived load management approaches can also be adopted using the WebGraph framework.

3.6 Personalization of Web Pages

The WebGraph framework allows and facilitates personalized services that are of high value and significance in the e-commerce environment. Based on the history of accesses and the cookie information, personalized graphs for different classes of users or individuals can be created. These graphs will be stored at the proxies and will be used upon access by the clients to disseminate data in a very effective manner. The attributes of the weblets and the edges would depend on the personal requirements of the individuals or the class. Several value-added services can be also considered that can be supported by the proposed framework.

Since the proxies also have limited access to some changes in the WebGraph, localized information such as local advertisements, schedule of local events, etc. can be integrated with the original web page. Similar approach is used in the contemporary television services, as mentioned earlier. Thus the personalization support can be adopted at different levels; for a single user, a group of users, or for a service locality.

3.7 Security Issues

As discussed in the previous section, the nodes as well as the edges can have attribute defining the level of security desired at the granularity of the weblets. Thus WebGraph facilitates the display of the same web page with different levels of security. This feature would be very useful for a variety of applications where one does not want to hide all information at the cost of security. A flexible security model is an attractive feature of the WebGraph. It should be ensured that the secure weblets are completely in control of the primary server and the attribute checking and changes of the secure weblets are handled only at the primary server.

4 WebGraph Implementation and Management

In the WebGraph framework, the graph for a page is a separate object, which can be sent by the server when the first request to the page is made. It can be updated periodically (push model) or can be pulled after a TTL parameter. The transfer of these objects can be done on demand or during idle periods. Graphs of popular pages may be pre-fetched and the updating can be planed during idle periods. Different weblets belonging to the same web page may have different TTLs. In addition, there could be correlation between the update versions of the weblets. A web page with some updated weblets and some stale weblets might produce a meaningless or incorrect presentation of the web page. We do address these issues in the proposed implementation.

The WebGraph framework can be used to support both pull and push models. In the pull model, the framework allows downloading of a subset of the weblets belonging to a page to be retrieved. The selection of the subset is based on the topology of the graph, and the node and edge attributes. The push model would be configured based on the rate at which the data is likely to change thus depending largely on the nature of the content. In the push model, weblets belonging to different subgraphs are downloaded periodically or on an event-driven basis.

One of the perceivable performance issues of WebGraph is the initialization overhead. Unlike CGI and other similar framework, there are usually multiple invocations for a dynamic request. If unscrupulously handled, this could counteract the performance gains of WebGraph. We thus propose an efficient implementation methodology for WebGraph.

In the current generation web interactions, a dynamic page request is usually served by one web application, either a CGI program, an ASP or JSP script, etc.. The application is usually an executable program or a script executable under some interpreters (Java interpreter, etc.). A dynamic request typically involves several heterogeneous tasks like computation, disk access, network-related processing or database access. To achieve parallelism among these tasks, either the programmers need to explicitly specify the concurrence or the underlying compiler/interpreter provide parallel scheduling within these tasks, either of which is not yet widely deployed in commercial systems.

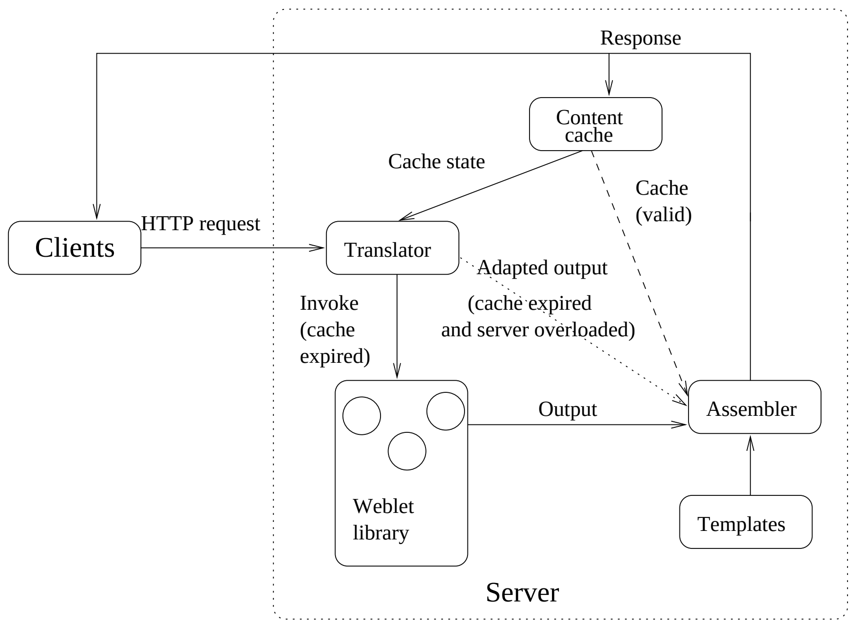

Figure 3: Implementation structure of WebGraph.

Figure 3 illustrates the implementation architecture for WebGraph. The translator accepts the dynamic request, checks the validity of the last response according to the dependency or validity conditions, loads the Weblets into its process space if their associated parts in the response page are stale. The template, together with the output from the weblets are fed to the assembler which is responsible for generating the final response and sending it back to the client.

When the translator loads the weblets, it assigns a thread to each of them and schedules them based on a scoreboad-like algorithm, i.e., any weblet can run regardless of their relative order in the template as long as their required resources are available. The exploitation of parallelism among weblets relies on the translator and resource specification of each weblet. The detailed design description follows.

4.1 Translator

Translator is the core of the WebGraph framework. From the server's perspective, it is the actual entity that serves the requests. It can be either an executable program or just a part of it. In the first case, it can be the sole executable program for all dynamic requests provided that the weblets exist in forms of loadable modules(explained below). The most obvious benefit to have a common server for all dynamic requests is that it improves responsiveness by not having to load executables from the disk each time. This concept is similar to the persistent residency feature of FastCGI [33]. After receiving a request, it looks up the cached response (if exists) and the graph that defines the page to decide which parts of the page need to be updated and invokes the corresponding weblets. There are two options to invoke weblets: load them into translator's process space or execute them in their own process space. In the first case, the weblets must exist in form of loadable modules, like Dynamic-Link Library (DLL) in MS Windows. The advantage of this method is that the overhead to load and initialize a weblet is reduced compared to the case of independent processes.

The translator is also responsible for admission control and load monitoring. Using this information along with the attributes of the nodes and edges, the translator provides the required level of QoS or the desired type of service. For example, if the e-commerce server of xyz.com shown in Figure 1 is highly loaded, the translator may not invoke the weblet to generate the radar image at all. Similarly, for a mobile client request, the translator may load only the weblets denoting coarse resolution images. The interface supporting these communication is discussed later in this section.

The translator is also capable of client identification for the purpose of security and personalization. Usually this is done via access authentication [28]. The request is accompanied with a cookie acquired during the previous visit for identification purpose.

4.2 Assembler

The function of the assembler is to create the web pages using the graph structure and the contents of the weblets. It gathers the output of the weblet that were computed for a request, replaces the stale weblets of the cached page according to the timing attributes of the graph, and recreates the entire page. It then sends the page to the client that requested it and also stores a copy in the server (or proxy) cache for future use. The assembler also updates the timing attributes (e.g. TTL) of the graph for the page.

4.3 Cache Organization

In this implementation, all the generated response will be cached in a memory pool maintained in the user-space for reuse purpose. The weblet output will be stored in the memory pool and not copied elsewhere. Caching of the history of responses imposes no additional resource requirements (e.g. memory capacity) compared to what is being used in the current generation web servers. The cache is organized in a hierarchical fashion. The root of the hierarchy is the root directory of URL's of all graphs. The graphs are in the second level with response headers, with pointers to weblet's output buffers. The hierarchical organization is illustrated in Figure 4.

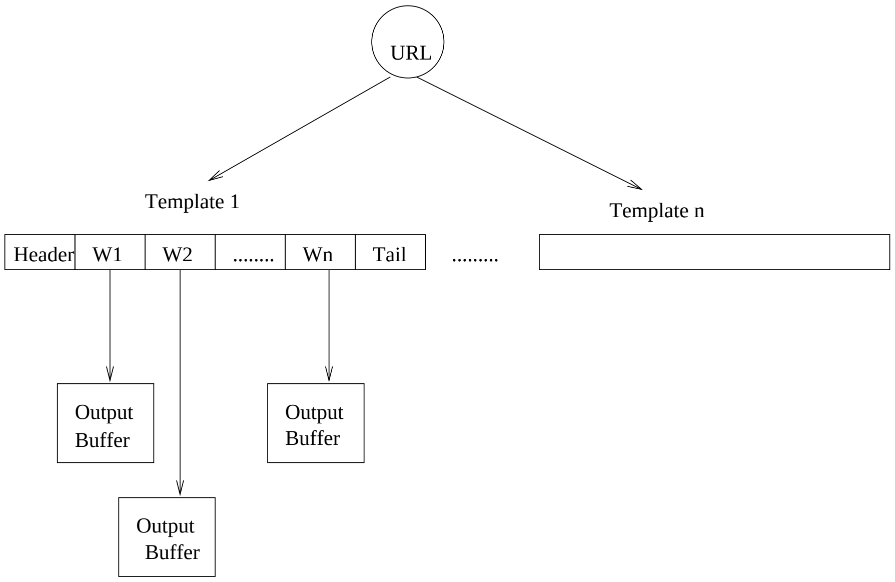

Figure 4: Web cache organization in the WebGuide framework.

4.4 Consistency & Management

In a distributed deployment of WebGraph framework, especially in server clusters, the issue of version consistency of weblet becomes an issue. Different weblets of the same web page residing on the same or different nodes must maintain their consistencies while getting updated. An updated web page should not have a combination of valid and invalid weblets. The update of nodes and links should be done simultaneously otherwise either stale or erroneous information will be replied to clients.

A version information in the weblet header is adopted for such purpose. The links in WebGraph also includes version information which is passed to and used by the weblets. The weblets check this with its own built-in version to see whether this call is valid. If a version conflict occurs, it either aborts the call or asks the translator for the updated version. We propose a two-phase upgrade of weblets to maintain the consistency. First the upgrade is done in the primary server. Then the primary server send an upgrade message to all other nodes that have a copy of the graph. Upon receiving this message, the proxy server nodes can either aggressively fetch the new weblet(s) or wait until a version conflict occurs.

5 Experimental Performance Study

We designed and implemented an experimental set-up using the methodology described in the previous section. The intent of the experimental study was to demonstrate the feasibility and examine the performance improvement that can be obtained through the WebGraph framework. Here we have focussed only on the performance improvement. Benefits of the other attributes can be also explored using appropriate workloads. However, most of the other benefits are obvious and can be justified as explained in the previous section.

The performance metrics measured in this study include *average response time* of successful connections which characterizes the user perceived latency, (the lower the better); *average throughput* of the server which indicates the processing rate, (the higher the better); *connection rate* which is the number of connections established per second, (the higher the better).

5.1 Configuration of the Experimental Set-up

The experimental set-up consists of a server, proxy, and a few clients. The server (733MHZ Pentium III) runs Apache 1.3.14 [4] for Microsoft Windows 2000 and the clients are four Sun UltraSparc machines executing modified WebStone 2.5 [5]that can send HTTP requests through a proxy. The proxy has the configuration same as that of the server. Apache is a widely used open source web server. WebStone is an open source, highly-configurable client-server benchmark tool for web servers. Using this tool, we can generate a configurable number of HTTP 1.0 GET requests for specific pages on a web server. Each of the clients send as many requests as possible using the assigned number of connections to the web server.

Figure 5 depicts the general topology of the testbed; each of the tests that we conducted used a little variation of the illustrated topology as indicated later. There are four heterogeneous servers with WebGraph functionality in the testbed. The *General Web server* is a relatively high end server. The *DB service access node* is the web server that can access backend database server, and the *Storage service access node* connects to a file server. The rationale for allocation for such access nodes is discussed in Section 5.2.3. The proxy server is placed between the server and the clients. All requests from the clients are directed to the proxy. The proxy either serves the requests or interacts with the server for serving the requests. The configuration is summarized in Table 3.

For our experiment, we considered an example web page as a representative of most dynamic pages. This web page consists of three different dynamic weblets with different attributes and functionalities as indicated in Table 4. The weblets are Microsoft Dynamic Linkable Libraries (DLL) written in C language. We have also examined other web page with various number of weblets. The overall results are very similar to that obtained using the example web page considered in this paper.

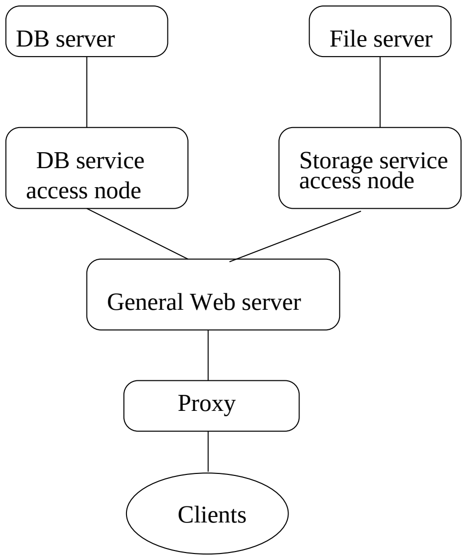

Figure 5: Testbed for experimental study.

Server	Configuration	Function
General web server	Intel PentiumIII 733MHZ, 128MB RAM	general computing, especially in CPU intensive tasks
DB service access node	AMD K6-2, 400MHZ,128MB RAM	processing DB query tasks
Storage service access node	AMD K6-2, 400MHZ,128MB RAM	processing file system intensive tasks
Proxy server	Intel PentiumIII 733MHZ, 128MB RAM	cache dynamic content, maintain WebGraph

Table 3: Configuration of the experimental set-up.

Weblet name	Function	Executing Server
A	DB query from a table with 16,000 records	DB service access node
B	Read files up to 400KB	Storage access node
C	Numeric calculation	General Web server

Table 4: Weblets of the example web page.

5.2 Performance Results and Discussion

As discussed earlier, using the capabilities of WebGraph, the performance of web accesses can be improved due to three reasons: dynamic content caching, parallel processing of the weblets, and load balancing at a granularity of weblets. In this section, we will demonstrate and quantify the performance improvements due to each of these features using the experimental set-up.

Three different types of tests were conducted using variants of the experimental set-up and the workload generated by the WebStone benchmark. In the first test, the proxy server is used to cache dynamic weblets and serve them based on the validity of their TTL attributes. In the second test, the performance comparison through parallel processing is examined with respect to a popular script language (PHP). PHP is the fastest-growing technology for dynamic Web pages [34]. It is growing at a rate of up to 15% each month, and is available on over 36% of Apache web servers. PHP provides better performance over other script languages [35]. We have used PHP 4.0 in our experiment. The third test investigates the performance improvement due to load balancing on the basis of partitioning the requests with respect to the desired functionalities. Furthermore, all the three tests validate the feasibility and functionalities of the WebGraph framework.

5.2.1 Performance improvement through dynamic content caching

In this experiment, a proxy server was placed between the general web server and the client cloud. The proxy was configured to cache both static and dynamic weblets based on their TTL attributes. The experimental configuration is shown in Figure 6. The proxy server caches weblets' output until their TTL's expire. If a request arrives that needs a weblet whose TTL has expired, the proxy server sends the request to the general server to retrieve the weblet and resets the TTL. Other weblets in the dynamic page whose TTL's have not expired still remain valid in the proxy server. In this test, we considered two different sets of TTLs for the three weblets: (15s, 10s, 5s) and (20s, 15s, 10s). We have also recorded the results with other TTL values, however, the behavior and trend of the results are very similar.

The performance comparison of the WebGraph framework is done with respect to a PHP server environment, which we consider as a respresentative of the contemporary web services. In the PHP-based server case, the proxy server does not caches any dynamic web content as is done in the current generation of proxy servers. All the dynamic contents are retrieved from the primary server. Figure 7 shows the variation of the server response time with respect to the number of concurrent clients. To fit the relatively high value of the PHP-based server response time, we have plotted the response time in logarithmic scale. It is observed that the response time of the server is considerably lowered by using the WebGraph framework. Higher TTL values will results in even lower response delays for the WebGraph-based servers. The corresponding variation of the server throughput is shown in Figure 8. It is clear from the plot that the throughput of the server is improved considerably by using the proposed framework.

The connection rate (defined earlier) at the server is measured and the results are plotted in Figure 9. It is observed that the connection rate with WebGraph is consistently better than that of the PHP-based server environment. Thus we infer that the response time, throughput, and connection rate, all are significantly improved by using the WebGraph framework.

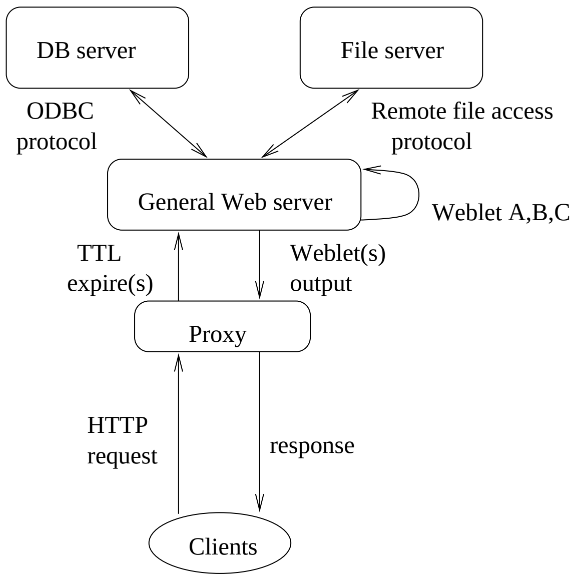

Figure 6: Dynamic content caching testbed.

5.2.2 Performance improvement through parallel processing

To the best of our knowledge, there is no server side script language or framework yet that can provide parallel processing among sub-tasks within a dynamic web page. The WebGraph framework facilitates the processing of the weblets independently in parallel. To examine the performance benefits, we again chose PHP as the base model server for comparison.

Table 5 presents the performance difference in terms of mean response time between the two schemes under the same workload. It is observed that the response time of the webpage in the PHP server is approximately the sum of its embedded individual weblets' execution time. This experimental result also acertains the claim that the PHP scripts run in sequential mode and cannot exploit the parallelism among sub-tasks. The WebGraph-based server executes the weblets in parallel, and displays the webpage when the last weblet gets executed. Thus the response time for retrieving a webpage from a WebGraph-based server will be approximately equal to the execution time of the slowest weblet. Overall, the response time observed in the WebGraph-based server is significantly lower than that of the PHP-based server.

Weblet	PHP Server	WebGraph Server
A	0.3s	0.3s
B	1.2s	1.0s
C	5s	1.4s
Web Page	6.2s	1.5s

Table 5: Response time comparison.

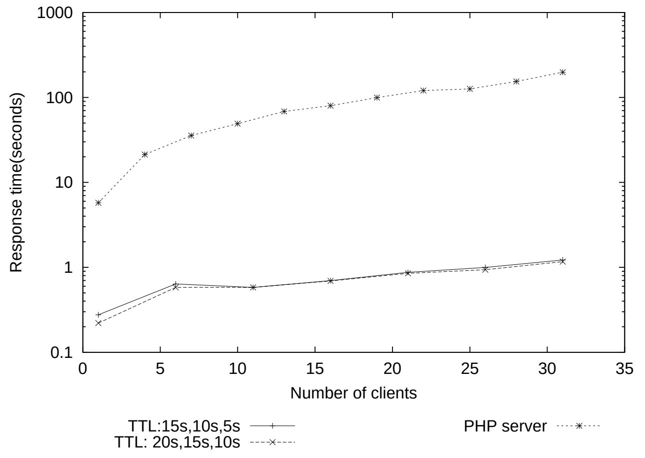

Figure 7: Server Response time with and without WebGraph.

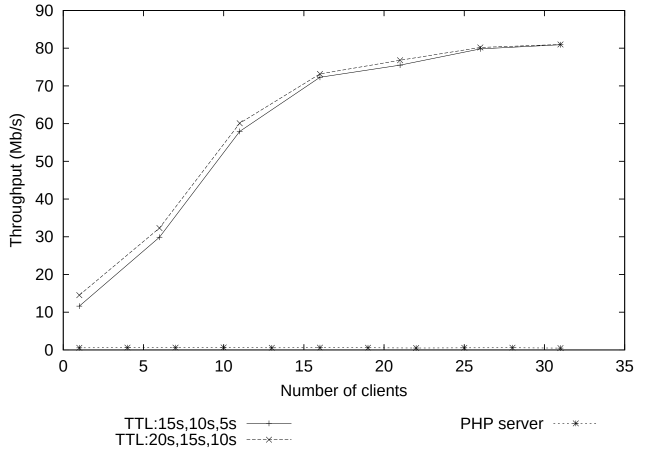

Figure 8: Server throughput comparison.

5.2.3 Performance improvement through load balancing

WebGraph can facilitate load balancing at weblet level and execute weblets on different servers. Unlike common practice in which all the servers are configured homogeneously and balanced both in CPU and peripheral devices, WebGraph enables heterogeneous configuration such that servers can be configured differently for their intended tasks. This also facilitate load balancing of tasks whose service access nodes are limited to DB service or storage service. The access nodes limitations may be due to software limitation(e.g. Database user license), security concerns, or geographical separation.

To evaluate and compare the performance benefits that WebGraph can facilitate due to effective load balancing, we set-up two experimental configurations as shown in Figures 10 and 11. Each of these set-up uses three nodes to behave as a server cluster. The traditional approach is shown in Figure 10, in which there is a dedicated load balancer, which distributes the incoming requests to the three web servers on the basis of their loads. These web servers are usually ho-

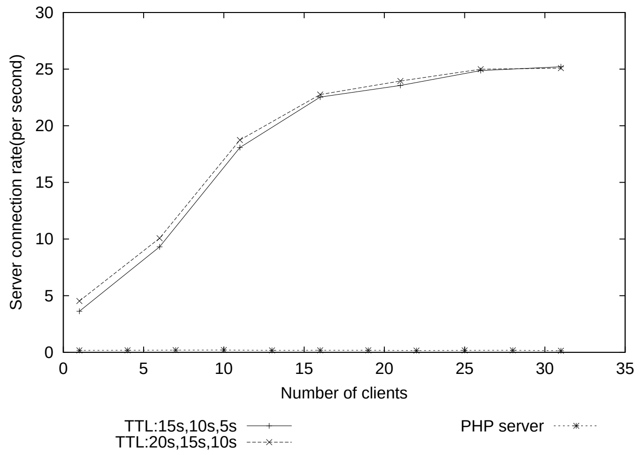

Figure 9: Server connection rate comparison.

mogeneous in terms of their capabilities and each one of them could access any of the backend servers.

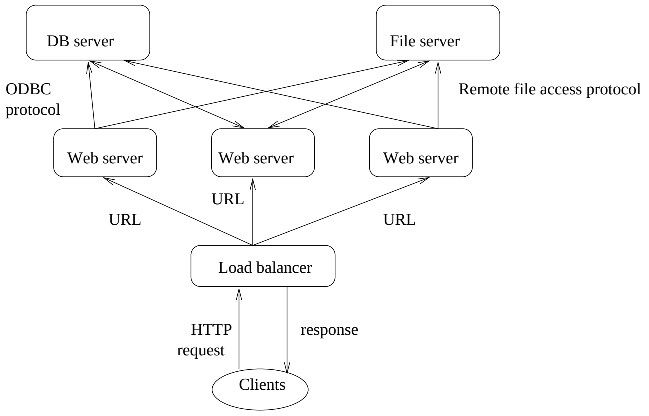

Figure 10: Load balancing testbedfor traditional server cluster.

The testbed for the WebGraph-based server cluster is shown in Figure 11. In this case a dedicated node (or a cluster of nodes) is used for accessing the backend servers. The dedicated nodes are heterogeneous and their capabilities are matched to the expected traffic to the corresponding back-end servers. The general web server receives all the requests and then directs the appropriate weblets to respective servers that act as service access points to the backend servers. Here we have shown one general web servers and one each for the service access nodes. However, there is no such restrictions and the number and size of these nodes could vary based on the load. The same arguement also applies for the traditional web clusters. For fair comparison, we have used three nodes for each of the testbeds. The load balancer is an extra component in the traditional server cluster.

The response time of the WebGraph-based cluster is much lower compared to the traditional web server cluster as evident from the plot shown in Figure 12. The rate of increase in response

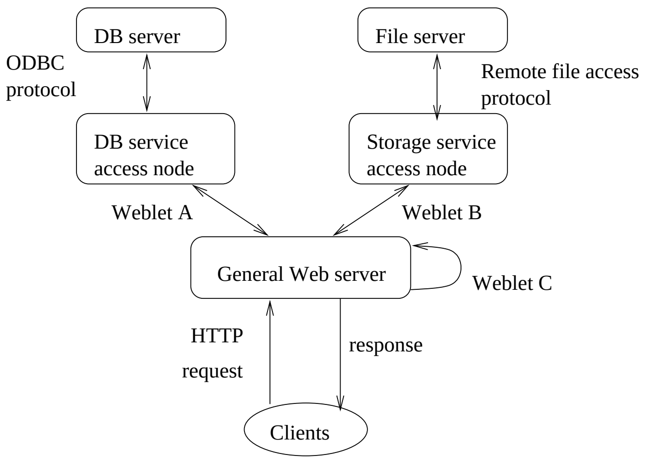

Figure 11: Load balancing testbed for WebGraph-based server cluster.

time with respect to the number of clients is much higher in case of the PHB-based server cluster.

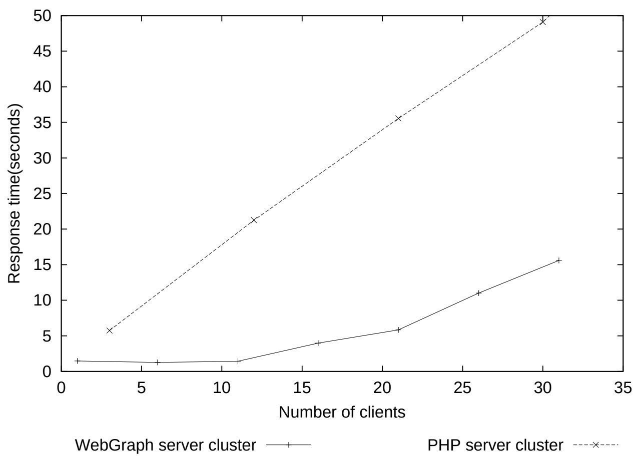

Figure 12: Response time comparison of the clusters.

The throughput comparison of the two configurations of server clusters is shown in Figure 13, and the corresponding rate of connections is plotted in 14. Both throughput and the connection rate are much higher for the WebGraph-based server cluster. As the number of clients increase beyond a certain number (16 in this case), the general web server or the load balancer gets saturated and thus the throughput drops. The throughput variations beyond 16 clients should not be compared as the clusters would be operating in an overload state. We have included that region in the graph to show the impact of overload.

6 Related Work

A significant amount of research efforts have been directed for improving performance of web servers. Several work has been reported on the web caching issues [8, 10, 11, 21, 15]. Frequently

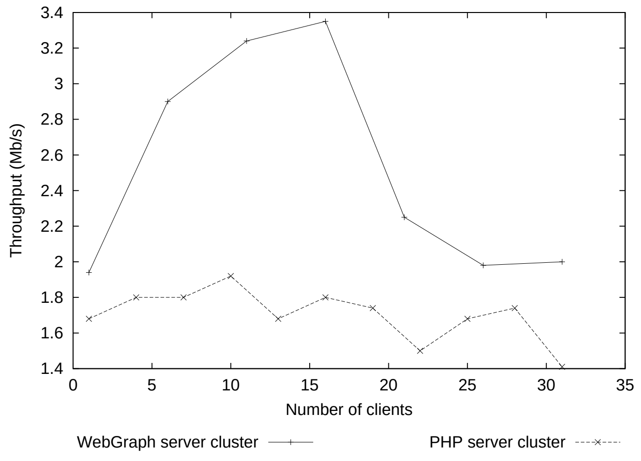

Figure 13: Server clusters throughput comparison.

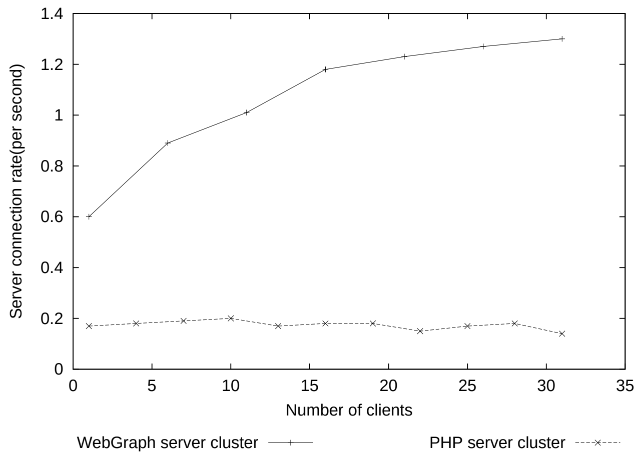

Figure 14: Server clusters connection rates.

accessed static web objects can be retrieved from the cache saving a significant amount of network bandwidth and delay. Effective replacement policies for web caching are discussed in [37, 16] Web caching, although effective for most static web sites, is not applicable for dynamic pages, which are usually not cached.

A control-theoretic approach for web caching is proposed in [31]. The primary idea is to provide QoS differentiation to web proxies based on performance metric, hit ratio. The most accessed the URL groups are treated with higher priority. Our work can work at a finer grained level.

Load balancing has been a very effective technique for managing load at server clusters. Mirror and replication of servers are been extensive used in the commercial environment. Several effective load balancing techniques have been proposed by researchers [7, 12, 14, 20, 22]. Overload control issues in web servers have not been adequately addressed yet. A content-based overload control technique was proposed in [1]. The service differentiation approaches proposed for web servers

[17] can be also used for load management. These work have targeted primarily for providing QoS support at the web servers.

There has been very limited work on addressing performance issues of dynamic requests. Cao et al. [13] have proposed the concept of active cache for caching dynamic contents on the web. The scheme allows servers to supply cache applets to be attached with the documents, and require proxies to invoke cache applets upon cache hits to furnish the necessary processing without contacting the server. The active cache approach may not be applicable for dynamic requests that need to access a back-end server. Further the active cache framework cannot provide all the additional features that WebGraph can provide as discussed in Section 3.

Caching of dynamic data has been studied in [29] in [19], where the work has focussed on rewriting the server applications to handle the insertion and deletion of cache items. The main goal in these approaches is the improvement of cache hit rates for dynamic requests. A publishing framework for web sites was proposed in [18] that uses a graphical representation of the web pages. However, the graph does not includes any attributes, and thus is used only for managing the publishing aspect. The works in [24] and [25] addressed dynamic page caching by separating static portions from dynamic portions and used special tags in the markup language to instruct the client which part can be cached and how to fetch dynamic content. A cooperative caching approach for dynamic data was proposed in [27] that is applicable in clustered environments only.

Edge Side Include (ESI) [26] is an ongoing industrial standard to facilitate dynamic content caching and assembly at the edge of the network. Its primary application is on surrogates, which are referred as edge servers in this paper. The surrogates can cache parts of dynamic pages and thus amortize the re-processing of valid contents and reduce origin servers' load. Currently, ESI is mainly intended for caching purpose and does not have as many applications as WebGraph.

Our work is different from all of the previous work on the performance of dynamic requests. Almost all of the earlier works have focussed on the caching behavior of the dynamic requests, which is just one aspect of the WebGraph. Unlike most of the earlier work, no client-side support is needed for WebGraph. Since we are targeting WebGraph implementation for proxies, the performance benefits would be for large groups of clients rather than programming individual clients and deriving performance benefits for them. In addition, as mentioned earlier, WebGraph can support and facilitate several other functionalities, which are equally important in several web applications environments.

7 Concluding Remarks

A new framework called *WebGraph* has been proposed in this paper for supporting dynamic as well as static contents in the web environment. The framework operates between the primary servers (the front-end as well as any back-end servers) and the proxies or edge servers. A graphical representation of the web with both node and edge attributes are stored for each of the web pages, which helps in rendering the web pages from the proxies. In addition to the performance in terms of lower response delay, WebGraph also facilitates caching, QoS support, overload control, load balancing, personalized services, and security. It provides a very flexible environment to managing and serving objects in the web environment. Experimental implementation and the results indicate the feasibility and the potential advantages of the framework. WebGraph can be also used to manage and provide contents to a heterogeneous set of clients (from heavyweight LAN users to lightweight mobile users) based on their resource availability. The framework would be a very useful tool in the e-commerce web services.

Acknowledgements

The authors would like to express their sincere appreciation to Krishna Kant from Intel Corporation for numerous discussions on the subject.

References

[1] T. Abdelzaher and N. Bhatti, "Adaptive Content Delivery for Web Server QoS," *International Workshop on Quality of Service*, London, UK, June 1999.

[2] Active Server Pages, http://msdn.microsoft.com/workshop/server/.

[3] Akamai Inc., http://www.akamai.com.

[4] Apache Group, http://www.apache.org.

[5] Mindcraft, "WebStone-The Benchmark for Web Servers", http://www.mindcraf t.com/webstone/

[6] V. A. Almeida and M. A. Mendes, "Analyzing the Impact of Dynamic Pages on the Performance of Web Servers," In *Computer Measurement Group Conference*, Anaheim, California, USA , December 1998.

[7] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra, "SWEB: Toward a Scalable World Wide Web Server on Multicomputers," In *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, Hawaii, USA , April 1996.

[8] M. Arlitt, R. Friedrich, and T.Jin, "Performance Evaluation of Web Proxy Cache Replacement Policies," Performance Evaluation, 39(1-4):149–164, February 2000.

[9] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable Content-aware Request Distribution in Cluster-Based Network Servers," Proc. of the 2000 Annual Usenix Technical Conference, San Diego, June 2000.

[10] O. Aubert and A. Beugnard, "Towards a Fine-Grained Adaptivity in Web Caches," In *Proceedings of the 4th International Web Caching Workshop*, San Diego, California , March/April 1999.

[11] G. Barish and K. Obraczka, "World Wide Web Caching: Trends and Techniques," IEEE Communications, May 2000.

[12] H. Bryhni, E. Klovning, and O. Kure, "A Comparison of Load Balancing Techniques for Scalable Web Servers," IEEE Network, July 2000.

[13] P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching Dynamic Contents (Objects) on the Web," In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, The Lake District, England , September 1998.

[14] V. Cardellini, M.Colajanni, and P. S. Yu, "Dynamic Load Balancing on Web-server Systems," IEEE Internet Computing, Vol. 3, No. 3, May/June 1999.

[15] G. Chandranmenon and G. Varghese. "Reducing Web Latency Using Reference Point Caching". In Proceedings of the IEEE Infocom 2001 Conference, April 2001.

[16] H. Che, Z. Wang, and Y. Tung. "Analysis and Design of Hierarchical Web Caching Systems". In Proceedings of the IEEE Infocom 2001 Conference, April 2001.

[17] X. Chen and P. Mohapatra, "Providing Differentiated Services from an Internet Server," Int. Conf. on Computer Communications and Networks, pp. 214-217, 1999.

[18] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed, "A Publishing System for Efficiently Creating Dynamic Web Content," *IEEE INFOCOMM 2000*, March 2000.

[19] J. Challenger, A. Iyengar, and P. Dantzig, "A Scalable System for Consistently Caching Dynamic Web Data," *IEEE INFOCOMM 99*, March 1999.

[20] L. Cherkasova, "FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service," In *Proceedings of the IEEE Symposium on Computers and Communications*, Antibes, France , July 2000.

[21] E. Cohen and H. Kaplan. "Refreshment Policies for Web Content Caches". In Proceedings of the IEEE Infocom 2001 Conference, April 2001.

[22] M. Colajanni and P. S. Yu, "Adaptive TTL Schemes for Load Balancing Distributed Web Servers". ACM Performance Evaluation Review, 25(2), September 1997.

[23] M. Colajanni, P. S. Yu, and V. Cardellini, "Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers," In *Proceedings of the 18th International Conference on Distributed Computing Systems*, May 1998.

[24] F. Douglis, A. Haro, and M. Rabinovich, "HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching," In *USENIX Symposium on Internet Technologies and Systems*, Monterey, California, USA , December 1997.

[25] F. Douglis and J. C. Mogul, "Potential benefits of delta-encoding and data compression for HTTP," In *Proceedings of the NLANR Web Cache Workshop*, Boulder, Colorado, USA , June 1997.

[26] Edge Side Include, http://www.esi.org.

[27] V. Holmedahl, B. Smith, and T. Yang, "Cooperative Caching of Dynamic Content on a Distributed Web Server," In *IEEE Symposium on High Performance Distributed Computing*, Chicago, Illinois , July 1998.

[28] HTTP/1.1, http://www.w3.org/Protocols/rfc2068/rfc2068.

[29] A. Iyengar, J. Challenger, "Improving Web Server Performance by Caching Dynamic Data," In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, December 1997.

[30] JavaServer Pages, `http://java.sun.com/products/jsp/`.

[31] Y. Lu, A. Sexana, and T. Abdelzaher, "Differentiated Caching Services; A Control-Thoeretical Approach". In Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, Ariz. USA , April 2001.

[32] J. C. Mogul, "Operating Systems Support for Busy Internet Services," In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, Orcas Island, WA , May 1995.

[33] Open Market, Inc. "FastCGI specification," `http://www.fastcgi.com`.

[34] PHP: Hypertext Preprocessor, http://www.php.net/.

[35] PerlMonth, `http://www.perlmonth.com/features/benchmarks/benchmarks.html`.

[36] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-Aware Content Distribution in Cluster-Based Network Servers," Proc. of the Conf. on Architectural Support for Programming Languages and Operating Systems, San Jose, CA Oct. 1998.

[37] K. Psounis and B. Prabhakar. "A Randomized Web-Cache Replacement Scheme". In Proceedings of the IEEE Infocom 2001 Conference, April 2001.

[38] T. Wilson, "E-biz bucks lost under SSL strain," Internet Week Online, May 20 1999. `htto://www.internetwk.com/lead/lead052099.htm`.

Author Biographies

Professor Prasant Mohapatra is currently an Associate Professor in the Department of Computer Science at the University of California, Davis. In the past, he was on the faculty at Iowa State University and Michigan State University. He has also held Visiting Scientist positions at Intel Corporation and Panasonic Technologies. Dr. Mohapatra received his Ph.D. in Computer Engineering from the Pennsylvania State University in 1993. His research interests include computer networks, Internet servers, and distributed systems. Dr. Mohapatra is currently on the editorial board of the IEEE Transactions on Computers and has been on the program/organizational committees of several international conferences. He is a Senior Member of IEEE.

Huamin Chen is Ph.D. candidate in the Department Computer Science, University of California, Davis. He received B.S. and M.S. from Huazhong University of Science and Technology, China in 1996 and 1999 respectively. His research interests are in Internet server performance improvement and QoS provisioning, computer networks and operating system optimization. Mr. Chen is a student member of IEEE.