CHAPTER 4
TRAFFIC ADAPTIVE OPTIMUM UPDATING SCHEME (TAOS)

4.1 INTRODUCTION
Web traffic increases with the increase in the growth of the web and its diverse characteristics. User behaviour and type of resources are the factors that contribute to the increase or decrease of traffic at a particular time. Secondly web does not impose any limit on the size of the resource, which implies that a text file may consist a few bytes or the contents of the book. The size of the resource has implications for the time required for the browser to download the content, and the load imparted on the network to transfer the data from one machine to another.

Web crawlers, the software agents of different search engines hold certain amount of network bandwidth to build and update the search engine repository in addition to the essential factors contributing to the web traffic. This thesis proposes an alternate solution to eliminate the needless requests of web crawlers during the updating process. This chapter describes the proposed updating scheme, its architecture and design issues.

4.2 SYSTEM DESIGN CHARACTERISTICS
Web is heterogeneous, dynamic environment where keeping track of the modifications of web pages distributed across boundaries is a complex task.
As the system becomes more complex, interconnected and diverse architects are less able to anticipate and design interactions among components, leaving such issues to be dealt at runtime. For example, in case of crawler based updating method the crawler component fails to interact with the web server of the web space regarding its load or check the load on network bandwidth before downloading a web document.

Traffic Adaptive Optimum updating Scheme (TAOS) aims to reduce the uncertainty in the process of updating the search engine repository. Autonomic computing based TAO updating scheme intends to accomplish up-to-date repository with optimum resource utilization. Autonomic computing (AC) is a computing environment with the ability to manage itself and dynamically adapt to change in accordance with system policies and objectives. Hence, AC provides self-configuring, self-healing, self-protecting and self-optimizing environments.

4.2.1 Autonomic characteristic of updating scheme

Web crawler method of search engine updating involves both internal and external factors of clients such as web page change, network load which are dynamic and not directly available to the search engines. TAOS controls the diverse interconnected components of the search engine repository (local database) and the global database by designing policies for optimum utilization of resources during the search engine repository updation. It uses autonomic computing to accomplish the task.
The proposed autonomic computing based updating scheme incorporates the self-configuring and self-optimizing characteristics effectively. Modifications of the global database are notified to the search engine repository considering the load of the web server and network bandwidth. Whenever the load on the server and the network bandwidth varies due to the burstiness in the user behavior the TAOS system configures not to notify the updates at that point of time.

The adaptive characteristics of TAOS will notify the updates considering the load on the resources. Moreover when a portion of the web page is modified, the TAOS system extracts the modified portion and notifies only that portion to the search engine repository. Notifying the updated portion alone will in turn help in optimal usage of the resources. TAOS concentrates to improve the freshness of the search engine repository with optimum resource utilization; self-healing and self-protecting characters are not used effectively in the proposed TAOS. They are considered to be concentrations of further research.

4.2.2 Delta encoding for updating scheme

Many HTTP requests cause the retrieval of slightly modified instances of resources for which the client already has a cache entry. Research has shown that such modifying updates are frequent, and that the modifications are typically much smaller than the actual entity. In such cases, HTTP would make more efficient use of network bandwidth if it could transfer a minimal description of the changes, rather than the entire new instance of the
resource. Transferring the instance of the resource is called delta encoding (Moghul et al, 2002). The sequence of generating HTTP message using delta encoding involves a number of different transformations on the body of value. The transformations supported by HTTP / 1.1 (Fielding et al, 2016) are discussed in the following section.

4.2.2.1 HTTP message generation sequence

HTTP /1.1 supports a number of different transformations on the body of a value:

**Content-coding** According to HTTP /1.1 Content-coding values indicate an encoding transformation that has been or can be applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information.

**Transfer-coding** According to HTTP /1.1 Transfer-coding values are used to indicate an encoding transformation that has been, can be, or may need to be applied to an entity body in order to ensure safe transport through the network. This differs from a content coding in that the transfer coding is a property of the message, not of the original entity”. Transfer codings are explicitly hop-by-hop transformations.

**Ranges** An HTTP client, using the range header, may request that the server return one or more subranges of the instance, rather than the entire instance value. HTTP/1.1 only supports byte_ranges, although there is some possibility that future extensions will allow for other kinds of range specifiers (such as chapters of a document).
A client signals its willingness to receive a content coding by sending an Accept-encoding header, listing the set of content codings that it understands. Optional message headers to support the use of instance manipulations are added in RFC 3229. A client signals its willingness to receive an instance manipulation by sending an A-IM (Accept-Instance-Manipulation) header, analogous to the set of instance manipulations it has applied using an IM (Instance Manipulation) header. A diagrammatic representation of the relationship between the transformations in order to conceptualise how delta encoding is applied to HTTP responses is given in figure 4.1.

<table>
<thead>
<tr>
<th>DATATYPE</th>
<th>OPERATION LEADING TO NEXT DATATYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
<td>choose acceptable variant, if needed</td>
</tr>
<tr>
<td>variant</td>
<td>apply content-coding, if any</td>
</tr>
<tr>
<td></td>
<td>compute/assign entity tag</td>
</tr>
<tr>
<td>instance</td>
<td>apply instance manipulation, if any</td>
</tr>
<tr>
<td></td>
<td>(delta encoding, range selection, etc.)</td>
</tr>
<tr>
<td>entity-body</td>
<td>apply transfer-coding, if any</td>
</tr>
<tr>
<td>message-body</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1 Diagrammatic representation of the relationship between the transformations in delta encoding (RFC 3229).

4.2.2.2 Requesting the transmission of deltas

RFC 3229 discusses the features that are needed as extensions in HTTP/1.1 in order to support the transmission of actual deltas. They include
1. A way to mark a request as conditional.

2. A way to specify the old instance, to which the client will apply the delta.

3. A way to indicate that the client is able to apply one or more specific forms of delta encoding.

4. A way to mark a response as being delta-encoded in a particular format.

The first two features are already provided by HTTP/1.1: the presence of a conditional request-header (such as "If-Modified-Since" or "If-None-Match") marks a request as conditional, and the value of that header uniquely specifies the old instance (ignoring the problem of last-modified timestamp granularity). The third feature, a way for the client to indicate that it is able to apply deltas (aside from the trivial 0% and 100% deltas), can be accomplished by transmitting a list of acceptable delta-encoding formats in a request-header field; specifically, the "A-IM" header. The presence of this list in a conditional request indicates that the client is able to apply delta-encoded cache updates. For example, a client might send this request:

```
GET /foo.html HTTP/1.1 Request method
Host: bar.example.net Headers
If-None-Match: "123xyz"
A-IM: vcdiff, diffe, gzip

Body Request body
```

Figure 4.2 Sample client request with instance manipulation headers (rfc 3229)
The above request explains that
- The client wants to obtain the current value of /foo.html.
- It already has a cached response (instance) for that resource, whose entity tag is "123xyz".
- It is willing to accept delta-encoded updates using either of two formats, "diffe" (i.e., output from the UNIX "diff -e" command), and "vcdiff".
- It is willing to accept responses that have been compressed using "gzip," whether or not these are delta-encoded.

If, in this example, the server's current entity tag for the resource is still "123xyz", then it should simply return a 304 (Not Modified) response, as would a traditional server. If the entity tag has changed, presumably but not necessarily because of a modification of the resource, the server could instead compute the delta between the instance whose entity tag was "123xyz" and the current instance.

4.2.2.3 Limitations of rfc 3229
Goals of delta encoding aim to ★ to reduce the mean size of HTTP responses, thereby improving the latency and network utilization ★ to avoid extra network round trips ★ to be fully optional for clients, proxies and servers and ★ to allow moderately simple implementations.

RFC 3229 does not include delta encoding of request messages or of responses to methods other than GET. Nothing of the specification specifically precludes the use of delta encoding for the body of PUT request.
In TAOS system the web page present in the web servers of the global database are the resources and the copy of the resource is present in the search engine repository. When a web page in the web server of global database is modified, the modified instance of the resource is much smaller than the actual page. The difference delta is extracted and sent to the search engine repository using the PATCH method. The modified instance of the web resource is sent as the body of the PATCH request.

4.2.2.4 PATCH method

Several applications extending HTTP require a feature to do partial resource modification. Existing HTTP functionality only allows a complete replacement of a document. The proposal (Dusseau, 2004) adds a new HTTP method, PATCH, to modify an existing HTTP resource. This specification defines a new HTTP 1.1 method to apply a delta encoding, or a "patch", to a HTTP resource. A new method is necessary to improve interoperability and prevent errors. The PUT method is already defined to overwrite a resource with a complete new body, and must not be reused to do partial changes. Otherwise, proxies and caches and even clients and servers may get confused as to the result of the operation.

The byte ranges are already used in HTTP to do partial downloads (GET method) as defined in rfc2616. However, they are not defined for uploads, and there are some missing pieces for uploads. For example, the HTTP specification does not define a particularly informative error to send if the byte range in a PUT is invalid. Byte ranges (or some other kind of range)
could be made to work in the specification but a more flexible mechanism (one that could also encompass XML delta encodings) was desired, as well as a method that would not confuse caching proxies. Reliable and tested delta encodings already exist, and this specification takes advantage of that existing work.

Some delta encodings for use in HTTP GET responses are defined in rfc 3229 (Moghul et al, 2002). That specification defines delta encodings for cache updates, not for user write operations. It does mean that servers can reuse delta-encoding algorithms to support both that specification and this proposal. PATCH specification defines a new method to alter a single existing resource, in place, by applying a delta encoding. A patch request body is modeled as a manipulation of an instance, where the instance would have been the entire document had it been PUT to the server, following the model of rfc3229 (Moghul et al, 2002). The operation is atomic. Note that WebDAV MOVE and COPY requests, if supported by the HTTP server, can be useful to independently rename or copy a whole resource before applying PATCH to either the source or destination URL to modify the contents.

A set of changes for a resource is itself a document, called a delta encoding. The delta encoding is uniquely identified through an instance manipulation as defined in rfc3229. Servers advertise supported delta encodings for PATCH by advertising these algorithms, and clients specify which one they’re using by including the name in the request. Not all instance-manipulations defined in the IANA registry are delta encodings; as of
October 2004, the instance manipulations which are also delta encodings are vcdiff, diffe, and gdiff. Servers should support PATCH and the vcdiff delta encoding for all authorable resources, that is all resources that support PUT.

Some requirements apply only to specific patch formats, and in PATCH specification those requirements are spelled out only for vcdiff.

4.2.3 Top down approach

TAOS system is designed to use top down approach instead of bottom up approach used by crawlers to update search engine repository. The top down approach has the advantage of having all the necessary knowledge already present for the program to use and thus it can perform relatively high-level tasks. In case of bottom up approach, the problem is tackled by starting with relatively simple abstract program. It is then designed to learn by itself building its own knowledge base and common sense assertions.

Web crawlers, which use the bottom up approach to update the search engine repository, build its knowledge base by learning the frequency of change and degree of change of the web document. The uncertainty in crawler-based updation is quantified based on the accuracy of the factors of frequency of change and degree of change of a web document. By designing top down approach, the uncertainty is eliminated and the change in the web document is notified based on the physical and logical factors. The knowledge required to notify the updates in an efficient manner are maintained in the nodes itself.
Despite the application of bottom up and top down approach to updation, both the approaches have a common problem of how to code commonsense computations or when to let the program learn for itself. The above problem is approached using an autonomic computing environment. Learning performed in AC environments eliminate the uncertainty due to requirement of knowledge about the remote client which the evolutionary computing, parallel processing techniques used in bottom up approaches does not support.

4.3 ARCHITECTURE

The architecture of a system has different dimensions at different levels. The conceptual view will help in understanding the wider application of the concept and the detailed techniques will explain the design in detail with the role of each component. The data structural view will brief the data structures used in implementing the concept. This section explains the conceptual view, detailed techniques and data structures of TAOS.

4.3.1 Conceptual view

TAOS system considers the web server of the global database as AC nodes. The AC nodes are designed to notify the updates based on the policies configured for optimal transmission. The centralized search engine server repository is also designed using autonomic computing and is referred as AC server. The AC nodes communicate with AC server based on the policy for which it is designed. To be precise the TAOS system is all about the policy management between the communicating resources (Fig. 4.3).
Since the AC nodes are autonomic in nature, it is designed to manage itself depending on the external parameters of user behavior, availability of bandwidth etc., of that particular AC node. Since the AC nodes are distributed around the globe, the nodes are designed to configure according to the web traffic of that particular region and at that particular time. This self-configuring and self-optimizing nature of the system will contribute much to the optimum resource utilization. Apart from self-configuring and self-optimizing, AC nodes transfer the modified instance of the changed resource adding to efficient resource utilization.

The AC nodes around the globe notify the updates to the AC server and receive the requests based on the policy design. It is designed to accept
the request based on various constraints such as popularity of the AC node, popularity of web page and size of the request etc. AC server demands processing of heavy traffic and therefore high end systems need to be deployed. AC server receives exponential requests from AC nodes of the database and the requests are then distributed to the cluster of the database according to the topic.

4.3.2 Detailed architecture

Web crawlers visit the web servers in intervals to identify whether there are updates in the web pages of server. If no web pages are updated, the crawlers' visit to the web server is considered unnecessary wasting the network resources. The TAOS aims to reduce the uncertainty in the process of updating of search engine repository. It aims to accomplish up-to-date repository with optimum resource utilisation.

The autonomic computing architecture of TAO updating scheme based on the generic architecture provided by Jana Koehler, Chris Giblin et al (2003) is given in Figure 4.4. Task knowledge of AC node contains representation of actual system behavior, the system itself and the environment as perceived by the system. When a new target behavior of updating is added to the task knowledge, which differs from the actual behavior, a deliberation process of extraction is triggered. The deliberation process extracts the updated portion, the delta and sends the sending delta
behavior to the web server negotiator component. The web server negotiator component allows issue of PATCH request depending on the bandwidth available or invokes predefined policies, which suits the status of the current
The web server execution component, which concentrates solely on executing behavior in a specific environment, issues a PATCH request.

The web server execution component communicates its status and results directly back to the task knowledge to which the request monitor is listening. The request monitor of the AC node collects the PATCH request status from the task knowledge and determines whether the requests have been correctly executed. The status results of any actual / target deviations are communicated to the failure recovery component of the AC node for recovery operation.

A PATCH request (Fig. 4.5) containing delta as the body is sent to the search engine repository, which is designed as AC server. The PATCH method requests that the request body be applied to the resource identified by the request URI. The request analyzer module of AC server scans the requests received from the AC nodes, prioritises the requests and sends to the task knowledge. It triggers the deliberation component to locate the URI of the request and sends applying delta behavior to the request analyser module. The request analyser module initiates the identifying behavior, which converts the URI to document identification code DocID to identify the web page in the search engine repository and the delta is applied. In order to authenticate the existence of web page in the search engine repository, the execution component has to verify that it is applying the delta encoding to a known entity. This is done with the help of E-Tags. E-Tags are response header fields which provide the current value of the entity tag for the
requested variant. The E-Tag is used in the If-Match header on the PATCH request to make sure the resource is still unchanged.

```
PATCH / file.txt HTTP/1.1
Host: www.example.com
Content type: text/plain
IM: vcdiff
If-Match: "e0023aa4e"
Content Length: 100

[vcdiff - bytes]
```

Figure 4.5 PATCH request

<table>
<thead>
<tr>
<th>Sync</th>
<th>Length</th>
<th>Compressed packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Length</td>
<td>Compressed packet</td>
</tr>
</tbody>
</table>

Packet (stored – compressed in repository)

<table>
<thead>
<tr>
<th>DocId</th>
<th>Ecode</th>
<th>URLLen</th>
<th>Pagelen</th>
<th>URL</th>
<th>Page</th>
</tr>
</thead>
</table>

Figure 4.6 Repository data structure

The delta encoded patch document is then applied to the URL. The execution component communicates the status and results directly back to the task knowledge to which the response monitor is listening. The execution component sends a response with the code of 204 that represents a status of no content. It also sends the content-MD5 header in response to PATCH
Fig. 4.7). This allows the client to verify the success of the operation. The response monitor of the AC server collects the PATCH response status from the task knowledge and determines whether the delta is correctly applied. The status response of any actual / target deviations are communicated to the failure recovery component of the AC node for recovery operation.

HTTP / 1.1 204 No Content
Content - MD% :Q2h1Y2sgw50
E-Tag : “e0023aa4e”

Figure 4.7 PATCH response

4.3.3 Data structures of TAOS system

TAOS system maintains number of data structures to keep track of the task knowledge of the AC nodes and AC servers, for scheduling the notification and requests of AC node and AC server. Queue structures are used in places where scheduling is applied. The main structures used in the TAOS system are shown in Fig 4.6.

4.3.3.1 Task knowledge of AC nodes and AC server

Autonomic computing systems maintain task knowledge of the actual system behavior, system itself and the environment perceived by the system. The system behavior records the user behavior and its impact to the resources in a particular time interval. That is, task knowledge of system behavior consists of the period when the network bandwidth and web server are overloaded in a day. When the resource availability is tested for delta notification, overload
period is checked and the delta is either notified or sent to the non-productive period queue.

Task knowledge of the system itself, which is the AC node, maintains the information related to the notified web pages. This is maintained to understand the age of the page, its frequency of change, which is used in study of freshness of the local repository. Task knowledge of the environment perceived by the system tracks the changes in the environment. It is used to check whether the notification of updates may overload the server or not.
Figure 4.8 Data structure of TAOS
The task knowledge of the system itself records the modified URL and its time of recent and last modifications. This helps in the study of calculating the freshness of the search engine repository. AC server receives the requests in exponential numbers and it transfers the requests to the cluster servers, which contains web pages. The task knowledge of the environment perceived by the system maintains the load of the cluster servers. The request analyser module checks the load on the clusters before sending the request.

4.4 SUMMARY

Autonomic systems will maintain and adjust their operation in the face of changing components, workloads, demands and external conditions and in the face of hardware or software failures, both innocent and malicious. They will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost. They will proactively seek to upgrade their function by finding, verifying, and applying the latest updates. The chapter discussed the autonomic characters supported by the proposed system. The chapter also discussed the components involved in TAOS, and the working of TAOS. Data structures designed to represent the knowledge of the system and knowledge perceived by the system to carry out the updation is also explained.