An Intelligence Layer-7 Switch for Web Server Clusters

Saeed Sharifian 1*, Mohammad K.Akbari 2** and Seyed A.Motamedi 3*

* Department of Electrical Engineering, Amirkabir University of Technology, 15914, Tehran, Iran
sharifian_s@aut.ac.ir
motamedi@aut.ac.ir

** Department of Computer Engineering & IT, Amirkabir University of Technology, 15914, Tehran, Iran
akbari@ce.aut.ac.ir

Abstract: In this paper we propose a new scheduling policy namely IRD (Intelligence Request Dispatcher) for web switches operating at layer-7 of OSI protocol stack in a web server farm. We classify dynamic and static requests. A hybrid Neuro-Fuzzy and LARD like approach is used to make route decision of dynamic and static requests separately. IRD select server with lower response time in an adaptive dispatching policy. In particular, we used the ANFIS (Adaptive Neuro Fuzzy Inference System) methodology to build a Sugeno fuzzy model to assigns each incoming dynamic request to the server with the least expected response time. This estimation is based on expected impact of each client’s individual request on server’s resources. This resources are, number of connection, CPU and disk usage. This parameters used to evaluate a load-weight of each server in a fuzzy manner and retrieved through feedback. For static requests an algorithm used to improve the cache hit rate in Web cluster nodes according to their load-weights. It’s goal is to improve load sharing in web clusters with dynamic and static information. A prototype implementation results confirm that proposed algorithm is more effective than representative request distribution algorithms, specially for dynamic requests that requires sophisticated policy and more service time than static information.

Key words: ANFIS, Web Server Cluster, web switch, Request dispatcher

1 Introduction

Given the exponential growth of internet caused the increasing traffic on the world wide web, it is difficult for a single popular web server to handle the demand from it’s many clients. Any single machine can easily become a bottleneck and single point of failure, the best way to cope with growing service demand is adding hardware resources instead of replacing it with faster one. In addition of cost, a single server machine can handle limited amount of request and cannot scale up with the demand. This will be clearer when we know 40% latency, as perceived by users, is causing on a web site side (Cardellini & al., 2002) . The need to optimize performance of web sites is producing a variety of novel architectures.

By clustering a group of web servers, it is possible to reduce the web server’s load significantly and reduce client’s response time when accessing a web document. Furthermore we can achieves high throughput and high availability too. Web servers in cluster work collectively as a single web resource in the network. Typical web server farms architecture includes a web switch that act as a representative for web site and provide a single virtual IP address that corresponds to web cluster. Web switches play an important role in boosting the web service performance. They acts as a centralized policy manager that receive the totality of requests and routes them among the servers of cluster. An important question is, how to allocate web requests among these servers in order to achieve load balancing and best performance.

Numerous dispatching algorithms were proposed for web cluster architecture that can divide in two major sub classes namely dynamic and static, static policies don’t consider any system state information and is less efficient than dynamic policies specially in CPU and disk consuming requests (Casalicchio & al., 2001) . However dynamic policies require mechanisms for monitoring and evaluating the current load on each server, gathering the information, combining them and taking real-time decisions. They need more decision time.

Internet traffic tends to occur in waves with intervals...
of heavy peaks, more over the service time of HTTP requests may have large variance (Crovella & al., 1997), (Feldmann & al., 1998). This bursts of arrivals and skewed service times, don’t motivate the use of sophisticated dynamic policies for all requested services, if the dispatcher mechanism has full control on client requests simple dynamic policies are as effective as their more complex dynamic counterparts for static requests (Casalicchio & al., 2001). A good web switch algorithm should support both simple and complex dynamic policies together to achieve best performance.

We propose an intelligent dynamic dispatcher algorithm to route client’s requests according to extracted information from request content’s and load status of each server. The algorithm executed on layer-7 web switch and classifies the clients requests to predefined classes of requests, if that class belong to static requests with small size, fast decision algorithm used for routing. Target server selection is according to simple policies to increase cache hit rate. If request belong to class of dynamic requests, which requires more service time, web switch use complex dynamic policy. Dynamic class requests route to the server with the least expected response time, which estimated for that individual request.

A Nero-fuzzy mechanism trained with different dynamic objects’ response time in different server load intensity, used in estimation mechanism, the request’s response time, estimated according to current server weight-load. Due to fluctuation of server workload distribution, the numerical data collected is vague. Fuzzy is an effective approach to utilize linguistic rule derived from numerical data pairs. It can be combined with powerful learning of past data in the form of Nero-fuzzy system in estimation mechanism of web server’s load-weight and response time (Jang, 1993). Currently, a lot of fuzzy estimation mechanism have been proposed. For estimation mechanism of expected response time, we use ANFIS with information from client request and server load status as inputs. One of the feature of ANFIS is adaptive learning mechanism that help the estimator to tune-up and refined. This adaptation is according to the cluster status and changing number and capacity of nodes. With combining simple dynamic policy for static requests and complex dynamic knowledge base policy for dynamic requests in IRD, we achieves precise and fast policy, That increase the performance of web cluster and minimize over all cluster’s response time. Implementation results confirms that IRD out perform other representative algorithms. In addition to robustness and adaptive nature.

The remainder of this paper is organized as follows. In section 2, we discuss some related work and their advantages. In section 3 we propose our new policy algorithm. Section 4 present estimation mechanism for response time and server load evaluation while in section 5, we describe a prototype of web switch operating at layer-7 that implements proposed algorithm and compares the performance with other policies. Finally we give our final concluding remarks in section 6.

2 Related Work

Various academic and commercial products confirm the increasing interest in web cluster architectures. In this architecture web switch as a front-end server uses one hostname and single virtual IP address for dozens of web servers and back-end servers. All HTTP requests reach the web switch, distributes among heterogeneous distributed web servers, that provide the same set of documents and services. Web switch is a key component in this architecture, according to the OSI protocol stack layer at which they operate, we may have layer-4 or layer-7 web switches (Schroeder & al., 2000). An example of this architecture illustrated in fig.1.

Layer-4 web switch that works at TCP/IP level, called content-blind, because they choose the target server when the client establishes the TCP/IP connection before sending out the HTTP requests. Since packets pertaining to the same TCP connection must be assigned to the same server node, the Web switch has to maintain a binding table to associate each client TCP session with the target server. The switch examines the header of each inbound packet and on the basis of the flag field determines whether the packet pertains to a new or an existing connection. Various policies algorithm can be implemented on layer-4 web switch range from static algorithms that don’t consider any system state information at the time of decision making to dynamic algorithms that either client-aware or server-aware or even combination of both. Client information that considered in decision are client IP address or TCP port. For servers the information can be load of server and number of active connections. Various kind of systems such as Magic router (Anderson & al., 1996), Distributed packet rewriting (Aversa & al., 2000), Cisco local director (Cisco, 2004), LVS (Ivs, 2004), IBM network dispatcher (Hunt & al., 1998), ONE-IP (Damani & al., 1997) and Dynamic feedback (Kerr, 2004) uses layer-4 web switch. One of frequently used algorithms in layer-4 web switch is Weighted Round Robin (WRR), that associates to each server a dynamically evaluated weight that is proportional to the server load state (Cardellini & al., 2002).

Layer-7 web switch can establish a complete TCP connection with the client and use OSI session layer
information, such as session identifiers, file size, file type and cookies (Aron & al., 1999), in addition to layer 2,3 and 4 information, prior to decision making. This kind of dispatching policy called content-aware because it extract information from HTTP requests before selection of target server. Some of client-aware algorithm working at layer-7 in web switch (Alteon, 2004), (F5 Networks, 2004) using static partitioning that assign dedicated servers for specific services. Although this method is useful from the system management point of view and achieve higher cache hit rates, but have poor server utilization. because resources that are not utilized cannot be shared among all clients. On the other hand, layer-7 routing introduces additional processing overhead at the Web switch and may cause this entity to become the system bottleneck. To overcome this drawback, design alternatives for scalable Web server systems that combine content and blind and content aware request distribution have been proposed in (Aron & al., 2000), (Song & al., 2000).

Such as layer-4 web switch decision making in this layer can be client or server aware, dynamic or static. Various kind of layer-7 algorithms exist, one of them is LARD (Pai & al., 1998). The LARD policy is a content based request distribution that aims to improve the cache hit rate in Web cluster nodes. The principle of LARD is to direct all requests for a Web object to the same server node. This increases the likelihood to find the requested object into the disk cache of the server node. We use the LARD version proposed in (Pai & al., 1998) with the multiple hand-off mechanism defined in (Aron & al., 1999) that works for the HTTP/1.1 protocol. LARD assigns all requests for a target file to the same node until it reaches a certain utilization threshold. At this point, the request is assigned to a least loaded node.

dynamic policies such as WRR and LARD work fine in Web clusters that host traditional static Web publishing services. In fact, most load balancing problems occur when the Web site hosts heterogeneous dynamic services that make an intensive use of different Web server’s components. Another frequently used layer-7 switch policy is Multi Class WRR (MC-WRR) or Client Aware Policy (CAP) (Casalicchio & al., 2001). The idea behind the CAP policy is that, although the Web switch can not estimate precisely the service time of a client request, from the URL it can distinguish the class of the request and its impact on main Web server resources.

In the basic version of CAP, the Web switch manages a circular list of assignments for each class of Web services. The goal is to share multiple load classes among all servers so that no single component of a server is overloaded. When a request arrives, the Web switch parses the URL and selects the appropriate server. CAP does not require a hard tuning of parameters which is typical in most dynamic policies. Because the service classes are decided in advance, and the scheduling choice is determined statically once the URL has been classified. Most recent layer-7 dispatching policy is Fuzzy Adaptive Request Distribution (FARD) (Borzemski & al., 2003). It classifies the client requests, and assigns each incoming request to the server with the least expected response time, estimated for that individual request. This policy that we used for basic development is the first algorithm that use fuzzy logic and separate neural network for response time estimation. FARD uses simple and inefficient Mamdani fuzzy model and membership function for estimation mechanism that have 40% to 70% estimation error (Borzemski & al., 2003). A sophisticated and centralized algorithm in FARD based web switch, decrease the maximum connection that web switch can handled. We use hybrid ANFIS approach that is combined fuzzy and neural network in a single estimator mechanism that have good performance in time series prediction and input-output mapping (Jang., 1993).

We use distributed ANFIS load estimator in each web server and gather server status in web switch. An intelligent algorithm in web switch make different routing decision according to type of request and server load. That outperform WRR, LARD and CAP algorithms. The Web switch cannot use highly sophisticated algorithms because it has to take fast decision for dozens or hundreds of requests per second. To prevent the Web switch becoming the primary bottleneck of the Web cluster, fast algorithms are the best solution. So we use fast routing algorithm for frequently used static requests.

3. Proposed Method

In web server cluster architecture that uses one way handoff (Aron & al., 1999) mechanism, each incoming URL request route to selected server, and the resulting resource is sent directly to client. The best selection of target server node is that minimize the response time for each individual user request (Borzemski & al., 2003).

We proposed an IRD (Intelligence Request Dispatcher) dispatching algorithm to make fast and intelligent knowledge based decision and minimize response time for requests. IRD uses client and server information to make decision. Each server in a web cluster has load-weight w to exhibit the server’s workload intensity, generally the load-weight of each server is determined by alternate web performance metrics that characterized the workload intensity in each time window, these important performance metrics can be explained as: server CPU load, server disk load and number of active connection. Due to the vagueness of these metrics, if expressed quantitatively, they can not illustrate server state in a precise way. In a highly dynamic system such as a Web cluster server, state information becomes obsolete quickly, So fuzzy approach is implemented to determine each parameter. Fuzzy inference system is based on fuzzy set theory and fuzzy rule-based approach in decision analysis and variety of fields, especially dealing with uncertain and complex systems.

We use ANFIS approach in each individual server node to extract load-weight w approximation from
explained metrics. Load weight produced by ANFIS from the crisp input values, given as percentage. More detailed defined in section 4 of the paper.

Client’s information are extracted from requested URL. The objects in server are classified to dynamic and static requests. Static requests contain HTML pages with some embedded objects like small picture files or large zipped files that can be cached. each static object is a file and can belong to class of certain size range.

Dynamic objects on the other hand can not be cached. content of dynamic requests is not known at the instant of a request and to be retrieved from web server, it can be as simple as sum of bill items that don’t intensively use server resource or can be complex. Dynamic content generated from data base queries which makes intensive use of disk resources is an example of complex dynamic request. Secure sites that use SSL protocol are other kind of dynamic requests that makes intensive use of CPU resources. Dynamic objects are treated individually and each one belong to a separate class of dynamic requests.

Block diagram of web cluster that uses IRD algorithm in web switch illustrated in Fig.2. it consist of load weight evaluator, object classifier, Static Small file Request Dispatcher (SSRD), and Dynamic and Large file Request Dispatcher (DLRD). Fig.3 show internal diagram of dynamic and large file request dispatcher. In heterogeneous clusters each server has its own corresponding model of response time estimator. A load weight evaluator daemon runs on each web server in the cluster.

The system works as follows. For the i-th incoming user request Ri, the web switch must select one of 1 to S web servers that called target server. All web switch decisions are independent of each other and serviced in order of arrival. If all of servers’ load exceeds a certain threshold, requests rejected to warranty QOS. Decision making to select target server is in the following way. When request Ri arrives, object classifier retrieve information about request from HTTP session, if it belong to small static objects, send it to small static object request dispatcher (SSRD) otherwise search the object lookup table for request specification. this specification for dynamic requests consist of file identifier that sorted according to service time. For large size static file that have been used rarely and have large service time in that is order of magnitude greater than small files (like zip files) we divide them to sub classes with certain size range. Each sub class, have an identifier sorted in lookup table according to mean service time of sub class. Part of this lookup table illustrated in Table.1.

### Table 1. A part of a lookup table

<table>
<thead>
<tr>
<th>ID</th>
<th>Dynamic objects</th>
<th>Large static objects</th>
<th>Initial Response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Php1</td>
<td>--------</td>
<td>1 ms</td>
</tr>
<tr>
<td>2</td>
<td>Php2</td>
<td>--------</td>
<td>1.3 ms</td>
</tr>
<tr>
<td>3</td>
<td>--------</td>
<td>500k&lt;size&lt;1M</td>
<td>1.7 ms</td>
</tr>
<tr>
<td>4</td>
<td>--------</td>
<td>1M&lt;size&lt;1.5M</td>
<td>2.1 ms</td>
</tr>
<tr>
<td>40</td>
<td>Php20</td>
<td>--------</td>
<td>0.5 second</td>
</tr>
</tbody>
</table>

The service time considered for sorting, is according to execution of each individual request on a idle web server. After extracting ID, it send to DLRD module. In DLRD an intelligence response time estimator REs exist for each individual web server S in cluster. When a new request arrived, each of S modules, uses it’s server weight load Ws, s=1..S of corresponding web server with file ID that retrieved from object classifier section and evaluate the response time TRs with ANFIS structure, assuming that this request is to be served by the given server. Next the comparator module chooses the target server as the one for which the estimated response time is minimal, let S denote the target server then request Ri forwarded to S-th server.

The system works in two mode, learning mode and high performance stable mode. In learning mode the system have adaptation capability and have extra module installed on each server S, that reports actual observed response time of requested object to web switch for training ANFIS module. Observed response time, is time between incoming and completing request. This observed time is in addition to objects’ ID send to target server’s response time estimator to use in adaptation mechanism. In learning mode maximum performance of web server cluster can’t achieved. On the other hand in high performance...
mode, all learning and adaptation process is finished and system can work without adaptation. This limit can reach if mean error between estimated and observed response time is lower than certain threshold and all object classes trained in all condition of load. If the incoming request belong to small static file, the corresponding dispatcher is used for selecting the target server $S$. A LARD-like algorithm used in SSRD, that increases RAM cache hit rate. When the object classifier distinguished the small static file, extract the URL and send it to SSRD unit. If this URL is requested for the first time then the SSRD select the target server $S$ that has lower load-weight $W_s$ in all web servers and forward the request to the selected server. If URL is known and used previously, SSRD forwards it to previously used server for that URL. If that server’s load-weight $W_s$ exceeds a certain threshold then request is forwarded to a server with lower load-weight $W_s$ in cluster. Because an URL is always forwarded to the same server, the server is very likely to have requested object in it’s RAM.

4. Estimation Mechanism for Response Time and Server Load

We use ANFIS approach for work-load evaluation and response time estimation. Adaptive Neuro-Fuzzy Inference System (ANFIS) is a class of adaptive networks that are functionally equivalent to Fuzzy Inference Systems (FISs) (Jang., 1993). Usually, the transformation of human knowledge into a fuzzy system (in the form of rules and membership functions) does not give exactly the desired response. So, there is a need to tune the parameters of the FIS to enhance its performance. The main objective of ANFIS is to optimize the parameters of a given fuzzy inference system by applying a learning procedure using a set of input-output data pairs (called training data). The parameter optimization is done in a way such that the error measure between the desired and the actual output is minimized. The architecture of ANFIS is a feed forward network that consists of 5 layers (Jang., 1993). Fig. 4 shows the ANFIS architecture for a two-input Sugeno-type fuzzy inference system used for response time estimation.

![figure 4. structure of ANFIS](image)

Each of server load-weight and object ID inputs of the ANFIS structures has five Gaussian membership functions, the rule base contains a total of 25 rules, which are as follows:

\[ f_i = p_i + q_i + r_i \quad i, j = 1, \ldots, 5 \quad (1) \]

where $p_i$, $q_i$ and $r_i$ denote the consequent parameters (Jang., 1993). The parameters of membership functions were obtained for each web server response time estimator by training the ANFIS structures with sufficient epochs. A combination of least-squares and back propagation gradient descent methods is used for training the fuzzy structures, in order to compute the parameters of the membership functions, which are used to model given set of inputs ($W_s$, ID) and single output $TR_s$. The computational detail of ANFIS structures is given below: Let the membership functions of fuzzy sets $A_i$ and $B_j$, be $\mu_{A_i}(W_s)$ and $\mu_{B_j}(ID)$, respectively in the form of (2)

\[ \mu_{A_i}(W_s) = e^{-\frac{(W_s - C_i)^2}{2 \sigma_i^2}} \quad i = 1, 5 \quad (2) \]

\[ \mu_{B_j}(ID) = e^{-\frac{(ID - C_j)^2}{2 \sigma_j^2}} \quad j = 1, 5 \]

where $\mu_{A_i}(W_s)$ and $\mu_{B_j}(ID)$ were chosen as Gaussian membership functions with the parameters of $C$ and $\sigma$. Tuning the values of these parameters will vary the membership function, which means a change in the behavior of the FIS. Parameters in this layer are referred to as premise parameters (Jang., 1993). In the second layer, the output of a node represents a firing strength of a rule. The node generates the output (firing strength) by multiplying the signals that come on its input in the form of (3)

\[ w_i = \mu_{A_i}(W_s) \mu_{B_j}(ID) \quad i, j = 1, 5 \quad (3) \]

The function of a node in the third layer is to compute the ratio between the $i$-th rule’s firing strength to the sum of all rules’ firing strengths according to (4)

\[ w_{ij} = \frac{w_i}{\sum_j w_j} \quad i, j = 1, 5 \quad (4) \]

$w_{ij}$ is referred to as the normalized firing strength (Jang., 1993). Every node in forth layer is a square node with a linear function $\eta_{ij}$ as,

\[ \eta_{ij} = w_{ij} (p_i W_s + q_i ID + r_i) \quad i, j = 1, 5 \quad (5) \]

where $p_i$, $q_i$ and $r_i$ are consequent parameters (Jang., 1993). The single node in fifth layer is labeled with $\Sigma$ which computes the overall output, TRs as the summation of all incoming signals, like (6)

\[ TR_s = \sum_{i,j} \eta_{ij} \quad s = 1, \ldots, S \quad (6) \]

During the learning process, the premise and consequent parameters are tuned until the desired response of the FIS is achieved (Jang., 1993). For the load weight estimation we use tree input ANFIS structure, with CPU load, disk load and number of active connection that normalized between 0 and 1, and each have five Gaussian member-ship functions. We use 125 rule base to cover all working area. The
output node produce the load-weight Ws in each server S, the server with larger Ws have more load. This structure provide a robust mechanism for load-weight estimation.

5. Implementation and Results
we implement a prototype web server cluster with a layer-7 web switch equipped with IRD, then compare proposed algorithm with three commonly used web switching policies, namely dynamic WRR, LARD and CAP. The Web cluster consists of a Web switch node, connected to the back-end nodes and the Web servers through a high speed LAN. The distributed architecture of the cluster is hidden to the HTTP client through a unique Virtual IP (VIP) address. Different mechanisms were proposed to implement a layer-7 Web switch at various operating system levels. The most efficient solutions are the TCP hand-off (Aron & al., 1999) and the TCP splicing (Cohen & al., 1999) that are implemented at the kernel level. TCP hand-off outperforms TCP splicing techniques (Aron & al., 2000) . The clients and servers of the system are connected through a switched 100Mbps Ethernet that does not represent the bottleneck for our experiments. The Web switch is implemented on a PC PentiumIV-2.3Ghz with 512MB of DDR 333 MHz memory as Web servers. All nodes of the cluster use a D-link DFE-538TX 100Mb/s network interface. They are equipped with a Fedora core2 Linux operating system (kernel release 2.6.5), and Apache 2.0 (Apache., 2004) as the Web server software.

On the Web switch node we used modified version of KTCPVS (Ivs, 2004) (Kernel TCP Virtual Server) as layer-7 kernel level web switch core, our IRD module implemented as user space module communicate with this kernel module. When new request arrives IRD receive URL from KTCPVS module and after estimation send back appropriate target server for that URL to kernel module. KTCPVS then route the request to target server.

We implement through on each web server a weight estimator module that collect information about server CPU and disk load and number of active connection and evaluate weights through ANFIS approach, every 1 second status manager module installed on the web switch gathers server load information through a socket communication mechanism. Another monitor module used in training mode that report actual response time between crating and destroying each apache thread in addition of its URL.

For benchmarking the web cluster we use 4 Pentium IV 2.3 GHZ as clients that are interconnected to the web cluster through a dedicated fast Ethernet. As synthetic workload generator we use Httperf 8.0 (Mosberger & al., 1997) benchmark. Each client prepared HTTP requests as fast as possible. To emulate a dynamic workload, we create two group of dynamic request stress both CPU and disk. they execute different complexity level query on MySQL5.0 (MySQL., 2004) database handled through ODBC mechanism on local nodes. A database had five tables, each of 10000 records and Each record consisted of three text, one integer and one float fields.

static files classified in to, two groups, small static files with size lower than 500KB, which consist total of 200 files in different kind. Large and rarely used static files have range from 500KB to 5MB which consist total of 50 files grouped in 20 classes according to their size. At the end of classification, the object lookup table have 40 entry for dynamic and large-rarely used static files.

In experiments we compare the performance of Dynamic WRR, LARD and CAP dispatching strategies under three different workload profile that illustrated in Table 2. static workload , light dynamic workload and intensive dynamic workload.

<table>
<thead>
<tr>
<th>profile</th>
<th>Small static</th>
<th>Large static</th>
<th>Stress CPU</th>
<th>Stress CPU&amp;disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>80%</td>
<td>20%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Light dynamic</td>
<td>60%</td>
<td>10%</td>
<td>30%</td>
<td>0</td>
</tr>
<tr>
<td>Intensive dynamic</td>
<td>40%</td>
<td>10%</td>
<td>20%</td>
<td>30%</td>
</tr>
</tbody>
</table>

We use homogenous server in our experiment. The main performance metric is the web cluster throughput measured in connection per second. First we determined the initial response time for each class of objects (40 classes) in lookup table, on an idle server, then sort this initial response time to obtain objects ID corresponding to response time. For each profile, working in learning mode is carried out for 30 minutes, and then we run our experiment in high performance mode. Fig 5 shows the system throughput in the case of static workload. We can see that IRD and LARD perform better than CAP and CAP is better than WRR.

Figure 5. Performance for static load

When we pass to consider a light dynamic workload in Fig 6, the performance results of IRD change respect to LARD & CAP, because dynamic requests stress the server resources in a quite different way.
best assignment of task to make load in balance at cluster. In addition simple policy like WRR is not efficient in dynamic loads. CAP is out perform than LARD because CAP has a dynamic nature compared to LARD.

![Figure 6. Performance for Light Dynamic Loads](image)

**Figure 6. Performance for Light Dynamic Loads**

![Figure 7. Performance of Highly Intensive Load](image)

**Figure 7. Performance of Highly Intensive Load**

Fig.7 presents the system throughput for an intensive dynamic workload that confirms that IRD outperforms LARD&CAP performance. The main motivation for this result is the better load balancing achieved by IRD with respect to LARD&CAP. An accurate and precise dispatching algorithm cause better load sharing and can served higher number of requests.

### Conclusion

In this paper we proposed a new intelligence layer-7 request dispatcher called IRD algorithm for dynamic load balancing in web server cluster that evaluate web server workload in a Neoru-fuzzy manner. On the other hand we classify incoming requests to cacheable and dynamic and use different policy for each of them. Especially for dynamic requests we use response time estimator based on ANFIS and assign the request to server with lower estimated response time. This algorithm can be trained and adapt to cluster. The results obtained from experiment show improvements in the cluster’s performance in term of connection per second. specially in heavy workload.

### References

- (Hunt & al., 1998) G.D.H. Hunt, G.S.


