A Multi-Class Load Balancing Algorithm (MCLB) for Heterogeneous Web Cluster

Kadiyala RAMANA[1]*, M. PONNAVAIKKO[2]

[1] Department of Computer Science and Engineering, SRMIST, Chennai, India

ramana.it01@gmail.com (**Corresponding author*)

[2] ProVost, Vinayaka Missions University, Chennai, India

ponnav@gmail.com

Abstract: Faced with increasing demand for network services by a huge number of users, requests to the web servers have significantly skyrocketed. Consequently, most of these servers need to run twenty-four hours a day, seven days a week with a high reliability and availability. Thus, the tremendous growth of the Internet has led the requirement of web server cluster management in order to deal with these issues effectively**.** Without using efficient mechanisms, an overloaded web server cannot provide great performance. In clusters, this overloaded condition can be avoided using load balancing mechanisms by sharing the load among available web servers. The existing load balancing mechanisms which were intended to handle static contents will be deprived from substantial performance under database-driven and dynamic contents. The most serviceable load balancing approaches to provide better results under specific conditions are Weighted Round Robin (WRR) and Client Aware Policy (CAP). By considering this, a Multi-Class Load Balancing algorithm (MCLB) was proposed for web server clusters and also an analytical model was proposed for calculating the load of a web server. The requests are classified based on the service time and keep tracking the number of outstanding requests at each webserver to achieve better performance. The service time of each request class type is used for load balancing. The experimental results demonstrate the effectiveness of the proposed approach by improving the average response time, error rate and throughput of the web server cluster.

Keywords: Load Balancing; Web Cluster; Content Aware; Throughput; Response Time.

1. Introduction

A web server system is the group of clusters in the networking environment to fulfill the web requests-based services. Through a web server system, any number of websites can be hosted for the purpose of education, business or technology. Due to intensive growth of website domain, the number of users is growing drastically day-by-day. As a result, the number of web requests has been arrived in an excessive manner at the web servers and it becomes overloaded and respond slowly. "In 1995, the number of internet users was lower than 1% in the world population, whereas today it is 40%. In 2016, there were 3.5 billion internet users while in 2005 there were 1.02 billion internet users" (Ramana & Ponnavaikko, 2015).

Whenever any web server responses are not up to the user's expectations, they get less interest on that website and then it affects the popularity and commercialization factor of that organizational website. All the web servers are required a suitable architecture to manage these excessive load of web requests. Several existing approaches for balancing the load of web servers are already working in the current environment, but still the web servers are responding slowly. Therefore, an efficient approach is required to balance the load of the web server system so that it can respond quickly and improve its availability to end users.

In web server cluster, load balancing includes a few noteworthy concerns. The essential concern is the estimation of work load. In various applications, workload has diverse meanings. In Internet services, the client's request is a basic building block of load balancing and its response with lively connections is a simple server load index.

Present web server clusters have some difficulties in providing services to the clients. First, in current websites dynamic workloads are becoming crucial, which imposes significant performance drop in web clusters with the shortcomings of present load balancing algorithms. When compared with the static web pages, the dynamic content requires high resource demands which lead to poor performance without suitable load balancing mechanisms in cluster-based web servers. Due to versatile demands, sometimes the request rate is greater than the cluster capacity. This is unpredictable with the flash crowds using the internet.

In this research paper, a dynamic and robust load balancing approach is proposed for content aware dispatchers. Three contributions are provided in the load balancing mechanism for web server clusters. The primary contribution is calculation of approximated load of a web server. Web requests are classified according to service time. The second contribution is a robust load balancing algorithm

https://doi.org/10.24846/v27i4y201808

named Multi-Class Load Balancing Algorithm. The final contribution is instigation of a web server cluster using the proposed load balancing mechanism. To estimate the effectiveness of the proposed algorithm some experiments are conducted and compared with some of the present algorithms. The investigational results prove that the proposed algorithm will provide substantial gains in error rate, throughput and average response time.

The paper is well-organized as follows: Section 2 catalogues some of the related works. Section 3 elucidates the proposed framework of web server cluster. Section 4 presents the multi-class load balancing mechanism. Section 5 gives the experimental results of the proposed algorithm. Section 6 outlines the conclusion.

2. Related Work

Eager et al. projected that the idea of load sharing was to increase the performance by reallocating the workload between the servers available in the system. Their work demonstrated that effortless adaptive load sharing strategies, which mount up extremely modest amounts of state information and uses in very simple ways produce noteworthy performance enhancements. It is also shown that particular enhancements do not take their toll on the monetary requirements. The paper concludes that in practice, simple policies provide the greatest potential, by reason of their mixture of nearly ideal performance and innate stability (Eager, Lazowska & Zahorjan, 1986).

Some of the presented works demonstrate that in order to administer web server clusters there is a need of load balancing algorithms, admission control and overload, performance optimization and architectural design, job dispatching and redirection mechanisms. So many algorithms are proposed for balancing the load in web server clusters. These algorithms are classified as content aware (layer-7) and content blind (layer-4) algorithms (Schroeder, Goddard & Ramamurthy, 2000; Andreolini, Colajanni &Nuccio, 2003).

2.1 Content Blind Algorithms

These algorithms are broadly divided into various subset of algorithms. Most popular approaches among those are Round Robin, Random Server Selection, Least Connection, Least Loaded, Weighted Round Robin, Request Counting, Weighted Least-Connection, Pending Request Counting and Weighted Traffic Counting. In case of heterogeneous servers, the requests are allocated to servers based on their constituted approximate capacities in WRR approach. The system administrator stipulates the percentage of requests to be dispatched to each server. Some numerous additional algorithms like Locality-based Least Connection, Source and Destination Hashing, Never Queue and Shortest Queue First, which are in need of out of the ordinary acquaintance to predict the best scheduling are discussed in a review paper (Ramana & Ponnavaikko, 2015).

2.2 Content Aware Algorithms

Pai et al. introduced content aware policy named Location Aware Request Distribution (LARD). To serve a request their paper has defined a set of servers and changed the set dynamically based on the active connections pending at the server. Their work also pioneered well-organized TCP Hand-off protocol which hand-off the incoming requests to the back end, after inspecting the content of the request by the front end (Pai et al., 1998). In LARD/R (LARD with Replication) when a subsequent same request comes in, it will be forwarded to the minimum loaded server among the servers (Ahn et al., 2004). Subsequently this approach does not differentiate between the types of various requests, if the server processes too many disk-bound or CPU-bound requests. This may cause load imbalance and performance deterioration of the whole cluster. These algorithms are pursued most of the time mainly for the reason that clients may be most of the times dependent on the content rather than other requirements.

To serve dynamic and secure web content, Casalicchio & Colajanni designed a mechanism named Client Aware Policy (CAP) by considering the content of incoming request, which obtains better performance (Casalicchio & Colajanni, 2001). The requests are categorized based on the impact the client's requests have on the server resources. This algorithm takes dispatching decisions according to the service type needed by the clients. In this algorithm the state of servers has not been considered. In cluster all servers will server all types of services.

In their work, Cherkasova & Karlsson proposed the Workload Aware Request Distribution which is a locality and content aware distribution policy that outlines most commonly accessed files. These particular files are accessed locally, by a server in the cluster, while the others are provided by different cluster nodes. To transfer a request from one server to another the Multiple Transmission Control Protocol (TCP) Hand-off has been used (Cherkasova & Karlsson, 2001).

Seo et al. introduced a set of prefetch algorithms where every node in the cluster will forecast the next web requests using access probability and the inter-reference time and prefetch the demanded objects from server local disks or the other back-end server nodes. By using Round Robin policy, a client session is allocated to a back-end server and a TCP connection is established among them for the complete session (Seo et al., 2008).

Sharifian et al. proposed a scheduling policy Intelligence Request Dispatcher, which uses Hybrid Neuro-Fuzzy and LARD to make a choice between serving dynamic or static requests. Central Processing Unit (CPU) and Disk usage are the metrics considered to assess a load weight of every server in a fuzzy method and regain it through feedback. Their paper concluded that this approach will improve the cluster performance in terms of connection per second especially in heavy workload (Sharifian, Akbari & Motamedi, 2005).

The researchers Pao & Chen projected a load balancing explanation by means of the remaining capacity of replicas to regulate how the next client' request should be accomplished. This enables the experts to estimate the behavior primarily to perceive the characteristics of the approach. The capacity is computed by means of available memory and CPU, the network transmission and the number of active connections pending at the server (Pao & Chen, 2006).

Zhang et al. proposed a Server Content based Queue (QSC) load balancing approach by classifying the web requests and considering the heterogeneity of web server (Zhang, Xiao-Ping & Yuan, 2010). In this approach, the client request is dispatched to the appropriate server which is least loaded. The load is calculated based on load state and server effectiveness. For each client's request, random distributing base probability was used for server load distribution to choose the appropriate server which depends on their weights. The selection course is carried out in a methodological approach such that there are no glitches during the processing.

Singh & Kumar proposed a web server queuing (WSQ) approach for improving the efficiency of the web server. Overloaded server can't provide best service. In this algorithm, load collector and status monitor are introduced as two new components, which compute the overloading condition of the web server. The investigation of current serving capacity of the web server is also achieved (Singh & Kumar, 2015).

2.3 Workload Classification

The workload measurement of web services agrees on the load balancing on the internet. One of the prevailing protocols of internet is HTTP which overrides TCP to carry the web traffic. Earlier studies on Web workloads found that some important characteristics like reference locality, file popular distributions, target file types, file size and client request patterns are common in the conventional information provider sites. When the requests are independent and same size random and round-robin strategies are good enough (Kwan, McGrath & Reed, 1995).

A lot of changes in web applications subsequent to vast developments have been made over the past two decades. For the majority part the most important change is the following: web page content is changing from static to dynamic leading to e-commerce becoming the foremost web application; and continuous media gaining interests. For users, dynamic pages will endow with a distant better experience than static pages, but they impose some additional overhead on server resources like Disk Input/Output (I/O) and CPU. Thus this may indulge in monetary problems. For existing load balancing techniques these changes in workload characteristics will impose a challenge. Some strategies are no longer pertinent as their versions and corresponding applications change day by day. As an instance, size-based strategy will not work for dynamic contents for the reason that of its unknown size, the service time is unpredictable (Harchol-Balter, Crovella & Murta, 1999). This is an inherent predicament in more or less all the types of dynamic techniques well-known in literature. Because of the dynamic page generations, the likelihood for caching the

requested files declines and some of the requested files are even non-cacheable. This has to be well addressed with proper experimental investigations and analyses such that this constraint can be worked out for a feasible elucidation.

Zhang projected novel load sharing policies in their research work (Zhang, 2000), which concerned with the efficient usage of both Memory and CPU resources. This research has paved the way for many fascinated researchers to pursue the policies and look for fruitful practical results through appropriate trialing. These policies accomplish high performance underneath Memory and CPU concentrated workload circumstances.

Zhang Xiayu et al. consider CPU, Memory, Bandwidth, Disk I/O and Buffer pool slice rate to compute the load index in a cluster. Their work employ the operation of extension set, matter-element theory and dependent function which are presented in extension theory (Zhang et al., 2007).

Qin et al. planned a load balancing approach considering CPU, Disk I/O/ and Memory resources to calculate the load. The Input Output Load Balancing (IOLB) algorithm provides better memory and CPU utilization under memory and CPU rigorous workload circumstances. This algorithm is able to deliver the similar level of performance as two already existing memory and CPU aware load balancing approaches (Qin et al., 2003).

Tiwari & Kanungo proposed a dynamic content aware load balancing approach for web cluster in heterogeneous environment. This algorithm uses utilization ratio, queue length and server's processing capability as load indices. As the content awareness is given importance in this work, the processing part is maintained stringently to augment the utilization ratio (Tiwari & Kanungo, 2010).

In their research paper, Saeed Sharifian et al. categorize dynamic requests into several different classes based on their impact on web server resources. The CPU usage is the most imperative basis of the tailback in the conception of dynamic contents (Sharifian, Motamedi & Akbari, 2011).

Most of the dynamic load balancing approaches evaluate the status of the load using periodic sampling in web servers. The most important issue in dynamic load balancing approach is web server load status accuracy. Several load balancing algorithms use CPU usage, number of active connections, disk usage and memory usage of web servers as load descriptors to calculate load status instantly (Schroeder, Goddard & Ramamurthy, 2000; Cardellini et al., 2001; Dahlin, 2000). The load status which is gathered from web server load monitoring, varies at various time scales and rapidly becomes obsolete (Penmatsa & Chronopoulos, 2007). So, the decisions made based on direct server resource measurements like CPU usage and mean web object response time as the load status, will be hazardous if not totally risky. In the proposed algorithm a queue model is used to calculate the load status from the online measurement of the queue parameters. The algorithm employs the information of file size and popularity distributions to depict mean response time to the available server resources.

3. Proposed Framework

In this research article, a heterogeneous web server cluster model is considered in which a set of servers are connected as illustrated in Figure1. In this cluster, servers are numbered 1, 2, …., n and the cluster is used to execute m classes of requests admitted by clients. Each server in cluster is composed of one exponential server having a service rate μ_i (i=1, 2, …., n), and uses First Come First Serve (FCFS) as its service policy where each request service rate equals μ_i/n if the i^{th} server has 'n' requests.

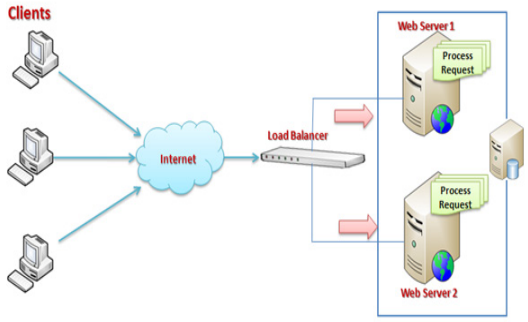

Figure 1. System Architecture

In heterogeneous environment the servers differ in terms of memory, disk space and speed. The load balancer is responsible for balancing cluster's workload and monitoring the available system resources. Figure 2 represents queueing model of the system. This contains request scheduling queue, request scheduler and n local request queues. The scheduler queue is a provisionally buffer sufficiently large to hold all

the incoming requests. The FCFS strategy is used for scheduling waiting requests in schedule queue, and local queues. It assurances fairness, does not need request's response time in advance, huge computing power and it is simple to implement. The load balancer is divided between two modules namely Supervisor and Dispatcher. For every request in j[th] class, the supervisor finds all servers in the cluster which satisfy the request requirements and group these servers in the set A_j. The supervisor then dispatches the set A_j to the dispatcher to select the server from A_j which provides the minimal predicted response time for executing the request. Clients can submit requests from any of the m classes to the server.

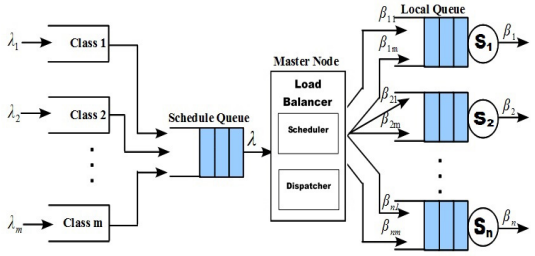

Figure 2. System queuing model

Table 1. *Notations and Definitions*

Notation	Definition
n	Server number (1≤n≤∞)
m	Classes of requests number admitted to the system (1≤m≤∞)
λ_i	j[th] class request arrival rate to the server
λ	Total server arrival rate from all classes (Equation 1)
P_{ij}	The probability in which requests from the j[th] class are forwarded to the i[th] server
β_{ij}	Processing rate (load) at i[th] node from j[th] class tasks (Equation 2)
β_i	Total processing rate (load) of server i. (Equation 3)
β	Server's total request processing rate (load) for all classes (Equation 4)
μ_{ij}	Allocated service rate at i[th] server for j[th] class of requests
μ_i	Total service rate of server i (Equation 5)
μ	Total Cluster service rate (Equation 6)
ρ_{ij}	j[th] class service utilization at the i[th] server (Equation 7)
ρ_i	i[th] server service utilization (Equation 8)
ρ	Cluster service utilization (Equation 9)

In this work the cluster's admitted requests are assumed to be completely autonomous and the requests are the computationally expensive ones. It was also assumed that requests of the j[th] (1⩽j⩽m) class reach to the cluster according to the ergodic process, like Poisson process, having identical, independent distributed interarrival times with rate λ_k. Synchronized arrivals are not considered.

The notations utilized throughout this work are represented in Table 1.

The server total request arrival rate from all the classes is denoted by λ and λ_i is the j[th] class request arrival rate to the server. Henceforth

$$\lambda = \sum_{j=1}^{m} \lambda_j \tag{1}$$

Denote β_{ij} as i[th] server load from j[th] class.

Let P_{ij} be the probability that i[th] server receives requests from the j[th] class, where $i = 1, 2, ..., n$ and $j = 1, 2, ..., m$.

Hence, the i[th] server workload from the j[th] class is calculated by

$$\beta_{ij} = \mathrm{P}_{ij}\lambda_j \tag{2}$$

where i = 1, 2, …,n j = 1, 2, …,m

So, the total workload of the i[th] server from all the classes can be expressed as

$$\beta_i = \sum_{j=1}^{m} \beta_{ij} = \sum_{j=1}^{m} \mathrm{P}_{ij}\lambda_j \tag{3}$$

As a result, the cluster's entire workload from all classes, β, can be calculated as follows

$$\beta = \sum_{i=1}^{n} \beta_i = \sum_{i=1}^{n}\sum_{j=1}^{m} \beta_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathrm{P}_{ij}\lambda_j \tag{4}$$

Denote μ_{ij} as the allocated service rate at the i[th] server for j[th] class request. So the corresponding predictab service time is computed by $\frac{1}{\mu_{ij}}$. Hence the i[th] server service rate can be computed by;

$$\mu_i = \sum_{j=1}^{m} \mu_{ij} \tag{5}$$

Consequently, the total cluster service rate is calculated by:

$$\mu = \sum_{i=1}^{n} \mu_i = \sum_{i=1}^{n}\sum_{j=1}^{m} \mu_{ij} \tag{6}$$

Denote ρ_{ij} as the j[th] class service utilization (traffic intensity) at the i[th] server. It is calculated by:

$$\rho_{ij} = \frac{\beta_{ij}}{\mu_{ij}} = \frac{\mathrm{P}_{ij}\lambda_j}{\mu_{ij}} \tag{7}$$

So, the service utilization of all requests assigned to i[th] server is computed by:

$$\rho_i = \sum_{j=1}^{m} \rho_{ij} = \sum_{j=1}^{m} \frac{\beta_{ij}}{\mu_{ij}} = \frac{\beta_i}{\mu_i} \tag{8}$$

In the same way, the total cluster service utilization is computed by:

$$\rho = \sum_{i=1}^{n} \rho_i = \sum_{i=1}^{n}\sum_{j=1}^{m} \rho_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\beta_{ij}}{\mu_{ij}} = \sum_{i=1}^{n} \frac{\beta_i}{u_i} = \frac{\beta}{\mu} \tag{9}$$

4. Multi-Class Load Balancing Algorithm

The algorithm for dispatching different classes of requests on multiple servers is presented in this section. The proposed algorithm considers the clusters computational resources to be heterogeneous and homogeneous. It balances the systems workload among servers by fairly distributing the service utilization in order to diminish the mean response time. In other words, the clusters workload is impeccably balanced between servers by making all the servers service utilization equal. This strategy minimizes the per-class mean response times. It involves two distinct decisions:

- The request allocation to the servers
- The request execution order at each server

The very first decision is considered an overall problem of load balance where requests are balanced between multiple heterogeneous servers to reduce the mean response time of each class. For the request allocation strategy the load balancer allocates arriving requests to servers instantly upon arrival in a probabilistic manner i.e. a request is dispatched to a server based on routing probability {Pij} $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$. The projected allocation algorithm finds that the response time for all classes is minimized. This algorithm selects the server that minimizes the average class response time, and should not be overloaded to compute the request at the same time. In order to achieve this, the proposed algorithm uses the following service utilization measure Li to detect the relative workload of each server i:

$$SLC_i = \frac{\rho_i}{\left(\frac{\rho}{n}\right)} \tag{10}$$

where ρ_i is the i[th] server service utilization given by Equation (8) and (ρ/n) is the clusters average server utilization given by equation (9). The proposed strategy aims to keep SLC_i very close to 1 which represents that the cluster's servers service utilization is fairly distributed.

Multi-Class Load Balancing Algorithm
Step -1 Publishing Phase (At Web Server)
Takes place periodically
1. For each web server, the load calculator calculates based on the equation (10)
2. Report the Server Load Capacity (SLC) to the Supervisor

Step-2 Selection Phase (At Dispatcher)
For each client request received from the classifier based on its class
1. Receive Load Capacity from all web servers periodically and initialize load values
2. Least Loaded Server, LLS_{Min}= Min (SLC_1, SLC_2, SLC_3, ……, SLC_n), where n indicates the number of available web server nodes.
3. The i[th] server such that $LLS_i = SLC_{Min}$ is considered as the least loaded server to process the current request.
4. Dispatch the request to the i[th] Server

Step-3 Processing Phase (At Web Server)
In i[th] server:
if (Number of requests in First In First Out (FIFO) request queue of i[th] server < queue capacity)
then
Add current request into FIFO request queue of i[th] server.
else
Drop the current request
else the i[th] server processes the current request.

The second decision is local at each server and consist in solving the best sequencing problem: given a various type of requests at a server, find the best queued request service order to minimize the average response time per class. It is assumed that the admitted requests to the cluster are totally autonomous and the requests are the costly ones. The FCFS strategy is used for queue applications. It guarantees definite fairness, requires no time to execute requests in advance, does not require enormous computer power and its implementation is easy (El-Zoghdy, 2012).

4.1 Workload Classification

Generally, web server serves different types of web objects, which are classified as static and dynamic requests. Static request mainly comprises of static data. This type of request is

the simplest one, which includes html or other unified documents. The static request service time is directly proportional to the file size, so the static requests are categorized based on the file sizes. The static request requires small amount of CPU demands.

To provide a rich set of web applications recent web sites supports dynamic contents. By using server side scripting languages like Java Server Pages, PERL and PHP or by using enterprise web applications like ASP.NET and EJB, the dynamic content is generated in dynamic requests. The dynamic contents cannot be cached completely, cannot be known in prior and they can be fetched from database and web servers.

As stated above, to better estimate the impact of every client's request on the web server load, the requests are classified into various classes based on the impact they have on server resources and on the service time.

Table 2. Classification of Requests

Class Type	File Name	Mean Service Time
C1	Home.html	100µs
C2	Load_balancing.pdf	10 ms
C3	Dynamicload.jsp	20 ms
C4	Loadbalancing.mp4	50 ms

4.2 Experimental Setup

The implementation of the investigational test entails software and hardware configurations as mentioned below.

4.2.1 Software Configuration

All the systems in the cluster run Windows 8.1 as their operating system. The web server uses apache v.2.3.9 configured with JSP 2.0 as the server-side scripting language. In order to avoid the connection rejections from server when many clients concurrently request services the extreme number of requests for every apache instance is increased up to 512. Because of this value the number of apache processes is never a limit on performance. MySQL v.5.1 is used as database server and configured with MyISAM table as non-transactional database tables, which provides higher performance in read only database interactions. Httpref tool is used to generate client workload for the experiments. Httpref is a synthetic workload generator and a web performance measurement tool. The load is varied on the web site by changing the number of simultaneous clients.

4.2.2 Hardware Configuration

The web server cluster consists of 20 systems. One system is used as the load balancer, and the remaining 19 systems are used as web server nodes. These nodes are Intel core i5 3.2 and 3 GHz CPUs with 8 GB, 4 GB and 2 GB of DDR RAMs. The load balancer node is an Intel core i5-4460 3.2 GHz CPU with 8 GB of DDR RAM. By using high-speed gigabit LAN switch these web server nodes are connected. Enough 3 GHz Intel core i5 systems are used as the client systems to guarantee that they would never become a bottleneck in any of the experiments. The distributed architecture of the web server cluster was hidden from the clients using a unique virtual IP address of the load balancer.

5. Experimental Results

In the subsequent sections, the outcomes of the experiments are explained with regards to average response time, throughput and error rate. Mostly, the approach which achieves high throughput, lower mean response time and lower error rate better uses the cluster resources and equally balances the load among servers. The parameters discussed are of main significance in order to understand the expected outcomes and to make the proposed algorithm a better one. Under overloaded circumstances, when a web server obtains a greater number of requests than its extreme capacity, the web server response time starts to vary and increases rapidly with the number of clients. In these experiments the low load and overload conditions are considered.

5.1 Mean Response Time

Figure 3 represents the average response time of web server cluster for the CAP, WRR and MCLB approaches under the generated workload. Under the low load circumstances, the mean response time is marginally lower for the multi-class load balancing approach when compared with other algorithms. To serve 3,300 client's requests the WRR approach takes 2 seconds, where as in the same mean response time the CAP approach serves 5,650 requests. At the same time, the multi-class load balancing algorithm served 7,400 clients in 2 seconds.

Due to the absence of an admission control mechanism, the overload condition starts at the dew-point, where the mean response time of CAP and WRR approaches becomes a bottleneck and increases aggressively. It can be determined that the proposed multi-class load balancing approach

improves the performance of the web server cluster to process more clients than the remaining two algorithms.

5.2 Throughput

Figure 4 represents the throughput of all the above mentioned three algorithms on the web server cluster with regards to the number of user's requests per second. Under light load conditions, the throughput raises linearly with the increasing number of client's requests.

The multi-class load balancing algorithm attains higher throughput to a certain extent when compared with the remaining two approaches. In multi-class load balancing algorithm, when serving 9,200 client's requests, the throughput reaches its peak value at the processing level of 1,294 requests per second.

For CAP algorithm, for 8,000 requests the saturation point is at 1,035 requests per second and for WRR approach for 5,000 clients it is at 616 requests per second. When one or more servers in the web cluster reach their dew-point is the primary reason for decrease in throughput for CAP and WRR approaches. When there are is greater number of client's requests in the web server, many of them will face time-out subsequently. Because of this, requests are not served by any of the server. This unexploited time is the reason for dropping in throughput of the web cluster, while the resource usage continues at 100%.

These outcomes clearly demonstrate that the load balancer of the multi-class load balancing algorithm works better than that of CAP and WRR algorithms. Also, under overload circumstances, the multi-class load balancing algorithm delivers constant throughput, while the other two approaches face bottleneck conditions and their throughputs of WRR and CAP are decreased. Finally, the conclusion is that the average request rate which can be served by the multiclass load balancing approach is about 1.24 times greater than with CAP and 2.08 times greater than with WRR.

5.3 Error Rate

Figure 5 shows the error rate of all the above mentioned three algorithms throughout the present experiments. After reaching the saturation point, the CAP and WRR algorithms error rates are increased sharply. Error Rate is a noteworthy metric because it measures "performance failure" in the application. It tells us how many failed requests are happening at a certain point during the time of load tests. In many load tests, this climb in Error Rate can be extreme. This speedy rise in errors shows the point, where the target system is stressed beyond its capability to deliver acceptable performance. Due to this, the error rate will increase and also the throughput of the server cluster will drastically decrease when the resource usage is 100%. Figure 5 illustrates the high error rates of WRR and CAP approaches with in accoradance with to the failed requests which require high service times. It can be determined that the multi-class load balancing approach improves the the capacity of the web server cluster by accepting an increased number of clients when compared with the remaining two algorithms and also maintains lower error rate.

6. Conclusion

In this research article, the problem of dispatching various classes of user's request on heterogeneous web server cluster is considered. A multi-class load balancing algorithm aiming to diminish the mean per-class response time is designed. The performance of the proposed multi-class load balancing algorithm is compared with that of the Weighted Round Robin (WRR) and Client Aware Policy (CAP) load balancing strategies. The investigational results prove that, the proposed dispatching strategy overtakes the WRR and CAP load balancing strategies with regards to average system response time, throughput and error rate. This particular improvement is obvious for moderate system workload. When the cluster workload is light or heavy, the performance of the three discussed dispatching strategies converges. In the future, an extension of the proposed multi-class load balancing strategy can be tested and investigated for more complex hierarchical models that replicate the real models.

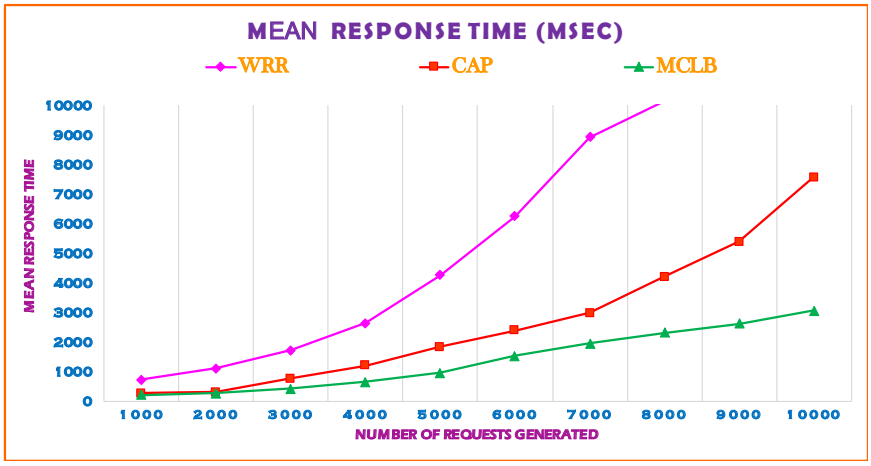

Figure 3. Mean response time variation Vs number of requests

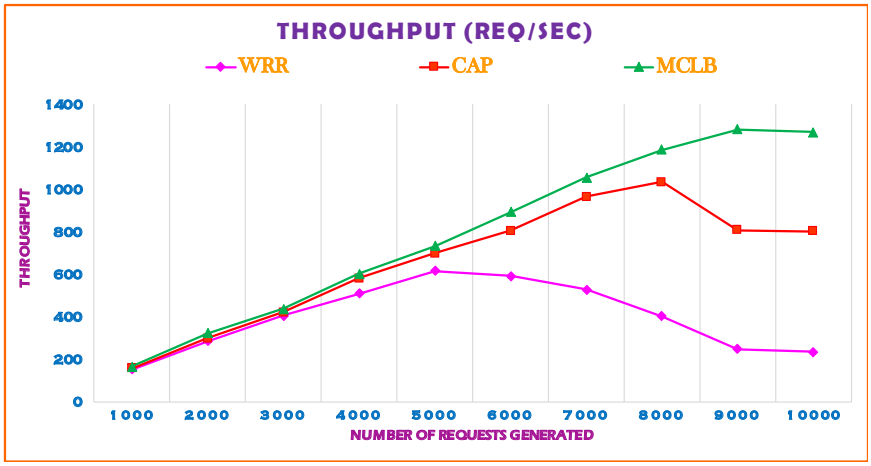

Figure 4. Throughput variation Vs number of requests

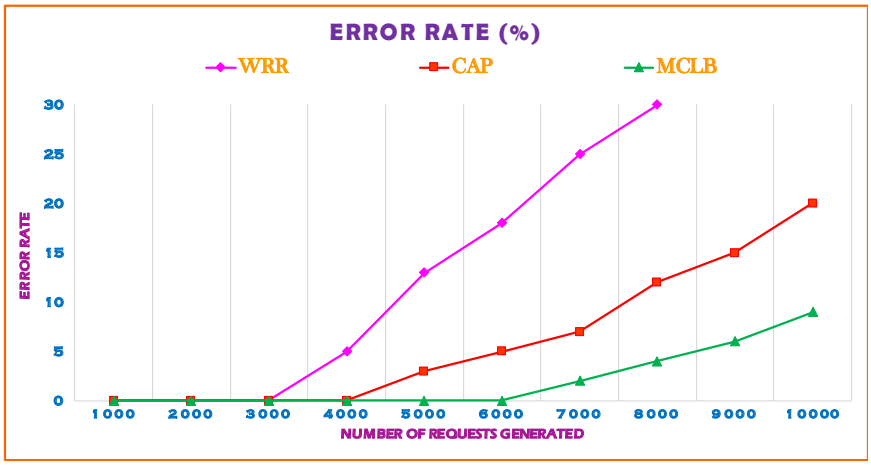

Figure 5. Error rate variation Vs number of requests

REFERENCES

1. Ahn, W. H., Kim, W. J. & Park, D. (2004). Content-aware cooperative caching for cluster-based web servers, *Journal of Systems and Software*, *69*(1-2), 75-86.

2. Andreolini, M., Colajanni, M. & Nuccio, M. (2003). Scalability of content-aware server switches for cluster-based Web information systems. In *The Twelfth International World Wide Web Conference* (pp. 270-280).

3. Cardellini, V., Casalicchio, E., Colajanni, M. & Tucci, S. (2001). Mechanisms for quality of service in Web clusters, *Computer Networks*, *37*(6), 761-771.

4. Casalicchio, E. & Colajanni, M. (2001, April). A client-aware dispatching algorithm for web clusters providing multiple services. In *Proceedings of the 10th international conference on World Wide Web* (pp. 535-544). ACM.

5. Cherkasova, L. & Karlsson, M. (2001). Scalable web server cluster design with workload-aware request distribution strategy WARD. In *Third International Workshop On Advanced Issues of E-Commerce and Web-Based Information Systems, WECWIS 2001* (pp. 212-221). IEEE.

6. Dahlin, M. (2000). Interpreting stale load information, *IEEE Transactions on parallel and distributed systems*, *11*(10), 1033-1047.

7. Eager, D. L., Lazowska, E. D. & Zahorjan, J. (1986). Adaptive load sharing in homogeneous distributed systems, *IEEE transactions on software engineering*, (5), 662-675.

8. El-Zoghdy, S. F. (2012). A hierarchical load balancing policy for grid computing environment, *International Journal of Computer Network and Information Security*, *4*(5), 1-12.

9. Harchol-Balter, M., Crovella, M. E. & Murta, C. D. (1999). On choosing a task assignment policy for a distributed server system, *Journal of Parallel and Distributed Computing*, *59*(2), 204-228.

10. Kwan, T. T., McGrath, R. E. & Reed, D. A. (1995). NCSA's world wide web server: Design and performance, *Computer*, *28*(11), 68-74.

11. Pai, V. S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W. & Nahum, E. (1998). Locality-aware request distribution in cluster-based network servers, *ACM Sigplan Notices*, *33*(11), 205-216.

12. Pao, T. L. & Chen, J. B. (2006, December). The scalability of heterogeneous dispatcher-based web server load balancing architecture. In *Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, 2006, PDCAT'06* (pp. 213-216). IEEE.

13. Penmatsa, S. & Chronopoulos, A. T. (2007, March). Dynamic multi-user load balancing in distributed systems. In *IEEE International Parallel and Distributed Processing Symposium, 2007, IPDPS 2007* (pp. 1-10). IEEE.

14. Qin, X., Jiang, H., Zhu, Y. & Swanson, D. R. (2003, August). Dynamic load balancing for I/O-and memory-intensive workload in clusters using a feedback control mechanism. In *European Conference on Parallel Processing* (pp. 224-229). Springer, Berlin, Heidelberg.

15. Ramana, K. & Ponnavaikko, M. (2015). Web Cluster Load Balancing Techniques: A Survey, *International Journal of Applied Engineering Research*, *10*(19), 39983-39998.

16. Schroeder, T., Goddard, S. & Ramamurthy, B. (2000). Scalable Web server clustering technologies, *IEEE Network: The Magazine of Global Internetworking*, *14*(3), 38-45.

17. Seo, E., Jeong, J., Park, S. & Lee, J. (2008). Energy efficient scheduling of real-time tasks on multicore processors, *IEEE transactions on parallel and distributed systems*, *19*(11), 1540-1552.

18. Sharifian, S., Akbari, M. K. & Motamedi, S. A. (2005). An Intelligence Layer-7 Switch for Web Server Clusters. In *3rd International Conference: Sciences of Electronic, Technologies of Information and Telecommunications SETIT* (p. 8).

19. Sharifian, S., Motamedi, S. A. & Akbari, M. K. (2011). A predictive and probabilistic load-balancing algorithm for cluster-based web servers, *Applied soft computing*, *11*(1), 970-981.

20. Singh, H. & Kumar, S. (2015). WSQ: web server queueing algorithm for dynamic load balancing, *Wireless Personal Communications*, *80*(1), 229-245.

21. Tiwari, A. & Kanungo, P. (2010, December). Dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness. In *Trendz in Information Sciences & Computing (TISC), 2010* (pp. 143-148). IEEE.

22. Zhang, L., Xiao-Ping, L. & Yuan, S. (2010). A content-based dynamic load-balancing algorithm for heterogeneous web server cluster, *Computer Science and Information Systems*, *7*(1), 153-162.

23. Zhang, W. (2000, July). Linux virtual server for scalable network services. In *Ottawa Linux Symposium* (Vol. 2000).

24. Zhang, X., Yu, Y., Chen, B., Ye, F. & Xingxing, T. (2007, November). An extension-based dynamic load balancing model of heterogeneous server cluster. In *IEEE International Conference on Granular Computing, 2007, GRC 2007* (pp. 675-675). IEEE.