RESEARCH ARTICLE OPEN ACCESS

A Survey on a Content Based Dynamic Load Balancing Algorithm for Heterogeneous Web Server Cluster

S.Tamilarasi [1], Dr. K.Kungumaraj [2]
Research Scholar [1], Mother Teresa Women's University, Kodaikanal
Head and Assistant Professor [2], Department of Computer Applications
Arulmigu Palaniandavar Arts College for Women, Palani
Tamil Nadu – India

ABSTRACT
In computing, load balancing distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process. Load balancing differs from channel bonding in that load balancing divides traffic between network interfaces on a network socket (OSI model layer 4) basis, while channel bonding implies a division of traffic between physical interfaces at a lower level, either per packet (OSI model Layer 3) or on a data link (OSI model Layer 2) basis with a protocol like shortest path bridging.
Keywords:- Load balancing, Work Load, Domain Name System, OSI Model

I. INTRODUCTION

The load balancing technology is widely used in current enterprise network to provide high quality and reliable service. Conventional load balancing technology is often achieved by specific hardware that is usually very expensive and lacks sufficient flexibility. Meanwhile, it is easy to become a single point of failure and would be restricted in virtualization environments. Thus, we propose a load balancing algorithm based on server running state, which can calculate comprehensive loading according to the CPU utilization, memory utilization, and network traffic of the servers. Furthermore, a load balancing solution based on software defined networks (SDN) technology is applied in this paper, and it is designed and implemented in Open Flow network. We combine network management and server state monitor in this scheme, in which the Open Flow switches forward the request to the least comprehensive loading server by modifying the packet.

Currently, traffic on the network is very huge and is growing rapidly. Network congestion and server overload are becoming severe problems faced by enterprises. In particular, the technologies and concepts such as cloud computing, virtualization, and big data make the issue particularly important. Typical enterprise networks are very complex, and with the growth of business, enterprises need to purchase more equipment, build more sophisticated networks, and handle more traffic.

[1] Most internet service providers use load balancing technology to assign the user's requests to different computers in data center. In order to minimize response time of requests and enhance user experience, requests from different users are processed by different computing nodes. [2]. Therefore, the amount of computation for each node is reduced. Load balancing technology is mainly for web service, FTP service, business-critical application, and other network applications [3]. Traditional load balancers are expensive, and the load balancer policy set needs to be formulated in advance, with its lack of flexibility leading to its

inability to deal with emergency situations. Traditional load balancer requires dedicated administrators to be maintained and does not allow users to design flexible strategies based on their actual network conditions. Since all requests are passed through a single piece of hardware, any failures on the load balancer will cause the collapse of the entire service. [4] Various load balancing schemes have some deficiencies in current situations. The fundamental reason is that the traditional design of the Internet has some problems. Under the impact of the new requirements, the bottleneck of traditional network architecture has been reflected in many aspects. People are looking for new options to meet changing business. Among many projects, SDN is the most influential and distinguished one, as the representative of SDN from the beginning, Open Flow. [5] has received wide attention from researchers. The goal of Open Flow is to change the way of controlling traditional network devices. In traditional networks, network devices forward data in accordance with distributed data management and, in the process of forwarding data from source to destination, individual equipment determines how to forward its data independently. Open Flow separates control module from the devices and puts it into an external server with a control program running on it; the server can send commands to the Open Flow switches to control the forwarding policy, and the control program can also provide an external application programming interface for network administrators to control the switch programmatically. This does not only reduce the need for manual configurations on switch, it can also provide greater flexibility in network management. Using load balancing technology in Open Flow network can well overcome some of the shortcomings of traditional load balancing and provide a simple and effective solution with high flexibility [6]. Due to the difference between the traditional Internet and Open Flow network, they inevitably differ from each other in using load balancing techniques in traditional network and Open Flow network. The traditional load balancing technology is not fully applicable to the Open Flow network. New problems have emerged, such as load balancing module design, the server operating status monitoring, and how to ensure the flexibility of load balancing.

On the other hand, many businesses in the enterprise network are migrating to virtual environments because virtualization technology can help companies to save money, consolidate servers, and maximize the utilization of limited resources. [7]. But now that load balancing technology is not mature enough in virtual environment, the application of traditional load balancing products is under restrictions in data center virtualization environments, which brought resistance to enterprise data center virtualization development. We proposed the design and implementation of Open Flow-based server clusters dynamic load balancing in a virtualized environment. The architecture not only is inexpensive but also provides the flexibility to write modules in the controller for implementing the customizable policy set. Internet applications can achieve real-time monitoring of load and timely access the appropriate resources by the flexible configuration capabilities of this architecture. An Open Flow switch can connect to multiple controllers in Open Flow we can use several servers as controller connecting to Open Flow Switch, so as to improve the robustness of the system.
[8] Tack vectors that subvert the defense with high probability. This has motivated research on trust management model

II. SERVER CLUSTER LOAD BALANCING METHODS

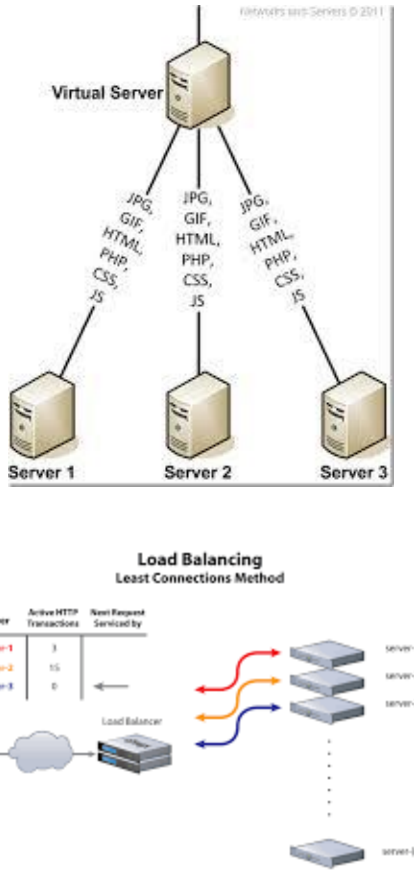


Figure -1.1

ALGORITHMS

3.1. Load Balancing Strategies

3.1.1. Random Algorithm

Immediately after each flow is forwarded to the controller, the controller randomly selects a server from the server list to process the client request. [15]

3.1.2. Round Robin

For each flow that is forwarded to the controller just a moment, the controller selects a server to process the client's request according to a certain order [16].

3.1.3. Server-Based Load Balancing Algorithm (SBLB)

(1)Dynamic feedback of the current server load mainly collects CPU occupancy rate, memory occupancy rate , and response time, but this information does not directly represent the load of a node, so it needs a function to convert these indicators and then get the load of the server :Due to the fact that there are different service types, which can have different influences on each part of the node, we introduced the parameter r; it is used to emphasize different degrees of influence of different parts.

(2)Processing computation ability of server node: when we compute load balancing, if the service nodes are heterogeneous, we not only should consider the node load but also must consider the node's processing capacity. For the A load-balancer in an infrastructure

The picture below shows how we usually install a load-balancer in an infrastructure: processing ability of node , it is mainly considered from the following several indicators: CPU quantity , CPU computation ability , CPU type, memory capacity , disk I/O rate , network throughput , and maximum process number (3)Calculating weight: when the server load balancing system comes into use, the administrator should set as initial weight to every server; as the server load changes, the weight will be adjusted. In order to avoid that the weight becomes a too large value, we set a range of weights to

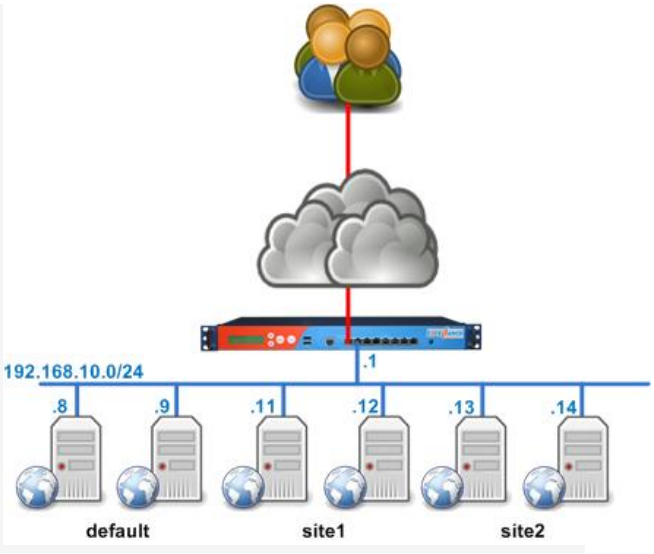


Figure-2.1

This is a logical diagram. When working at layer 7 (aka Application layer), the load-balancer acts as a reverse proxy.

So, from a physical point of view, it can be plugged anywhere in the architecture:

- in a DMZ
- in the server LAN
- as front of the servers, acting as the default gateway
- far away in another separated data center

6.2 Load Balancing Algorithm

The following algorithm will show the load balancing by making use of review matrix and load matrix. This algorithm will consist of three functions namely sort ( ), load balance ( ) and balance ( )

REVIEW MATRIX: R (Server ID, Present Memory, Processing Speed and Load Status)
LOAD MATRIX: L (Cluster ID, Memory Limit, Processing Limit and Load Status)
Sort ( ) will sort the web servers in the Review matrix.

```
SORT ®
{
for(i=0;i<n; i++)
{
for(j=i+1;j<n; j++)
{
if(R[i][1] > R[j][1])
{ Swap; }
else if(R[i][1] == R[j][1])
{ if(R[i][2]>R[j][2])
{ Swap; }
} } } }
```

Load Balance ( ) will check for load status of all web servers. If load status is uneven, it will call balance (cluster[i], job, j) to balance load within the cluster.

```
LOAD_BALANCE(R[n])
{
SORT®;
for ( i=1 ; i <= n ; i++)
{
for ( j=0; j < cluster[i].no servers; j++)
{
if (load status = = uneven)
{
Flag=Call BALANCE (cluster[i] , job ,
j);//BALANCE WITHIN
} } } }
```

Balance ( ) will be executed for all web servers within the clusters as well as among the clusters. It will re-assign the uneven load status job to any of the web server where the job.memory and job.processing <= cluster.job and cluster.memory respectively.

If none of the web server load is even, balance ( ) will check it for rest of the clusters.

```
BALANCE (cluster[i] , job , j)
{
for ( k = j ; k < cluster[i].no_servers ; k++)
{
if ( job.memory <= cluster[i].WS[k].memory &&
job.processing <= cluster[i].WS[k].processing)
{
Assign job to this server and update R;
Return(0);
}
else
BALANCE(cluster[i++],job,0); // BALANCE
AMONG CLUSTERS
} }
```

II. PROBLEM ISSUES

The state-of-art web applications communicate and coordinate with number of geographically dispersed information resources providing information to huge number of users. Homogeneous server clusters are not capable of satisfying the growing demand of such applications including real time audio and video, PHP, JSP, ASP etc. Heterogeneity involves handling of low level interoperability issues e.g. mismatch of hardware, operating platforms, programming languages, database schema, topology etc. Scalable server cluster allows addition of new servers as the load increases without disrupting the services. Moreover, it also provides better reliability by

gracefully transferring the load from server which is unavailable due to failure or for preventive maintenance. Heterogeneity with scalability makes the system more complex. The existing dynamic load balancing (DLB) algorithms are not directly applicable for distributed scheduling in such environments. In this paper, we propose a DLB algorithm for scalable heterogeneous server cluster using content awareness. The algorithm considers server's processing capability, queue length, utilization ratio etc. as load indices. As the cluster supports multiple services, at the primary level, we have used content awareness forwarding algorithm and at the secondary level, waited round robin algorithm has been used.

III. PROBLEM FORMULATION

The web servers of popular websites often need to be based on distributed or parallel architecture while preserving a virtual single interface. This will result into small latency time and less burden on each server. Different websites use different strategies to distribute load among web servers but most of the schemes concentrate on only one factor that is number of requests, but none of the schemes consider the point that:

• Different type of requests will require different level of processing efforts to answer.

HTTP is not a connected protocol: it means that the session is totally independent from the TCP connections.

Even worst, an HTTP session can be spread over a few TCP connections… When there is no load-balancer involved, there won't be any issues at all, since the single application server will be aware the session information of all users, and whatever the number of client connections, they are all redirected to the unique server. When using several application servers, then the problem occurs: what happens when a user is sending requests to a server which is not aware of its session? The user will get back to the login page since the application server can't access his session: he is considered as a new user.

To avoid this kind of problem, there are several ways:

- Use a **clustered web application server** where the session are available for all the servers
- **Sharing user's session information** in a database or a file system on application servers
- Use IP level information to maintain **affinity** between a user and a server
- Use application layer information to maintain **persistence** between a user and a server

NOTE: you can mix different technique listed above.

Building a web application cluster

Only a few products on the market allow administrators to create a cluster (like Web logic, tomcat, jboss, etc…). I've never configured any of them, but from Administrators I talk too, it does not seem to be an easy task.

By the way, for Web applications, clustering does not mean scaling. Later, I'll write an article explaining while even if you're clustering, you still may need a load-balancer in front of your cluster to build a robust and scalable application.

Sharing user's session in a database or a file system. This Technique applies to application servers which has no clustering features, or if you don't want to enable cluster feature from.

It is pretty simple, you choose a way to share your session, usually a file system like NFS or CIFS, or a Database like MySql or SQL Server or a memory cached then you configure each application server with proper parameters to share the sessions and to access them if required.

I'm not going to give any details on how to do it here, just Google with proper keywords and you'll get answers very quickly.

IP source affinity to server:

An easy way to maintain affinity between a user and a server is to use user's IP address: this is called Source IP affinity.

There are a lot of issues doing that and I'm not going to detail them right now (TODO++: an other article to write).
The only thing you have to know is that source IP affinity is the latest method to use when you want to "stick" a user to a server. Well, it's true that it will solve our issue as long as the user use a single IP address or he never change his IP address during the session.

Application layer persistence:
Since a web application server has to identify each users individually, to avoid serving content from a user to an other one, we may use this information, or at least try to reproduce the same behavior in the load-balancer to maintain persistence between a user and a server.

The information we'll use is the Session Cookie, either set by the load-balancer itself or using one set up by the application server.

What is the difference between Persistence and Affinity:
Affinity: this is when we use information from a layer below the application layer to maintain a client request to a single server.
Persistence: this is when we use Application layer persistence IE: "balance round robin" or "balance least connection" persistence IE: "balance round robin" or "balance least connection"

- Status record of all the web servers that are associated with one domain name must be considered.
- Mechanism to handle a situation when one of the servers is not working.

IV. RELATED WORKS

[1].A Dynamic Load-Balancing Algorithm for Heterogeneous Web Server Cluster

As increasing the embedded objects and the database searching tasks in Web Pages, there is larger difference among the loads of different server in a cluster system, which becomes more difficult for a heterogeneous Web server cluster to achieve high performance. In this paper, the authors present a dynamic load balancing algorithm MDC(Multiplicative Decrease in Critical area). For each of the servers in the cluster, the algorithm can more accurately evaluate the current load state by using the Equivalent Load Alternant and can more efficiently restrain the occurring of the reject service phenomenon by using a special MDC operator. Besides, the authors apply a method of random distributing base probability to assign each request to an appropriate server in terms of their weight. All the parameters that will be used in the algorithm can be acquired by simulated test. The authors also provide improved approximation results of above algorithm for the case where documents consist of relatively many embedded objects or database searches and lots of requests arrived the dispatcher synchronously.

[2].Workload-aware load balancing for clustered Web servers

We focus on load balancing policies for homogeneous clustered Web servers that tune their parameters on-the-fly to adapt to changes in the arrival rates and service times of incoming requests. The proposed scheduling policy, ADAPTLOAD, monitors the incoming workload and self-adjusts its balancing parameters according to changes in the operational environment such as rapid fluctuations in the arrival rates or document popularity. Using actual traces from the 1998 World Cup Web site, we conduct a detailed characterization of the workload demands and demonstrate how online workload monitoring can play a significant part in meeting the performance challenges of robust policy design. We show that the proposed load, balancing policy based on statistical information derived from recent workload history provides similar performance benefits as locality-aware allocation schemes, without requiring locality data. Extensive experimentation indicates that ADAPTLOAD results in an effective scheme, even when servers must support both static and dynamic Web pages.

[3] A content-based load balancing algorithm with admission control for cluster web servers

With the growing demands for web-based applications, cluster web servers emerged as a reliable and leading resource in internet infrastructure. Managing performance of the cluster web servers under heavy load conditions is a critical task specifically when it comes to the advent of dynamic contents and database-driven applications. In this paper we propose a new load balancing algorithm namely IQRD (Intelligent Queue-based Request Dispatcher) for web-switches of the cluster web servers which operates at layer-7. The IQRD aims to achieve better load balancing with the help of request classification, performance isolation and dynamic remaining capacity estimation mechanisms. For this, a queuing model was employed for each class of requests in each node of the cluster to provide an estimation regarding the node remaining capacity. This value is used as a load descriptor (index) in the load balancing algorithm and also used by the admission control mechanism. The implementation results with synthetic and realistic workloads confirm that IQRD effectively balances loads among servers in the cluster and achieves better response time and throughput compared to other load balancing algorithms. However, the IQRD algorithm offered more processing overheads both in the web-switch and the web servers, but presented a better load balancing among web servers, even when the request rates were beyond the cluster capacity.

[4] EquiLoad: a load balancing policy for clustered web servers

We present a new strategy for the allocation of requests in clustered web servers, based on the size distribution of the requested documents. This strategy, EquiLoad, manages to achieve a balanced load to each of the back-end servers, and its parameters are obtained from the analysis of a trace's past data. To study its performance, we use phase-type distribution fittings and solve the resulting models using a new solution method for M/PH/1 queues that only requires solution of linear systems. The results show that EquiLoad greatly outperforms random allocation, performs comparably or better than the Shortest Remaining Processing Time and Join Shortest Queue policies and maximizes cache hits at the back-end servers, therefore behaving similarly to a "locality-aware" allocation policy, but at a very low implementation cost.

[5] Energy conservation in heterogeneous server clusters

The previous research on cluster-based servers has focused on homogeneous systems. However, real-life clusters are almost invariably heterogeneous in terms of the performance, capacity, and power consumption of their hardware components. In this paper, we argue that designing efficient servers for heterogeneous clusters requires defining an efficiency metric, modeling the different types of nodes with respect to the metric, and searching for request distributions that optimize the metric. To concretely illustrate this process, we design a cooperative Web server for a heterogeneous cluster that uses modeling and optimization to minimize the energy consumed per request. Our experimental results for a cluster comprised of traditional and blade nodes show that our server can consume 42% less energy than an energy-oblivious server, with only a negligible loss in throughput. The results also show that our server conserves 45% more energy than an energy-conscious server that was previously proposed for homogeneous clusters.

[6] A content-based dynamic load-balancing algorithm for heterogeneous web server cluster

According to the different requests of Web and the heterogeneity of Web server, the paper presents a content-based load balancing algorithm. The mechanism of this algorithm is that a corresponding request is allocated to the server with the lowest load according to the degree of effects on the server and a combination of load state of server. Besides,

apply a method of random distributing base-probability to assign each request to an appropriate server in terms of their weight. All the parameters that will be used in the algorithm can be acquired by simulated test. Experimental results suggest that this algorithm can balance the load of web server clusters effectively, make full use of the existing source of software and hardware, highly improve the server's performance, and even make the best use of the web server.

[7] Power optimization for dynamic configuration in heterogeneous web server clusters

To reduce the environmental impact, it is essential to make data centers green, by turning off servers and tuning their speeds for the instantaneous load offered, that is, determining the *dynamic configuration* in web server clusters. We model the problem of selecting the servers that will be on and finding their speeds through mixed integer programming; we also show how to combine such solutions with control theory. For proof of concept, we implemented this dynamic configuration scheme in a web server cluster running Linux, with soft real-time requirements and QoS control, in order to guarantee both energy-efficiency and good user experience. In this paper, we show the performance of our scheme compared to other schemes, a comparison of a centralized and a distributed approach for QoS control, and a comparison of schemes for choosing speeds of servers.

Building a scalable web server with a global object space support on heterogeneous clusters.

Clustering provides a viable approach to building a scalable Web server system. Many existing cluster-based Web servers, however, do not fully utilize the underlying features of the cluster environment, and most parallel web servers are designed for homogeneous clusters. In this paper, we present a pure-Java-implemented parallel Web server that can run on heterogeneous clusters. The core of the proposed system is an application-level "global object space", which is an integration of the available physical memory of the cluster nodes for storing frequently requested objects. The global object space provides a unified view of cluster-wide memory resources, and allows transparent accesses to cached objects. Using a technique known as cooperative caching, a requested Web object can be fetched from a node's local memory cache or a peer node's memory cache to avoid hot spots and excessive disk operations. A preliminary prototype system has been implemented by modifying the W3C's Jigsaw Web server. We obtained good speedups in the benchmark tests, indicating that clustering with cooperative caching can greatly improve the performance of a Web server system.

V. WEB CLUSTERS

5.1 Architecture

A Web cluster refers to a Web site that uses two or more server machines housed together in a single location to handle user requests. Although a large cluster may consist of dozens of Web servers and back-end servers, it uses one hostname to provide a single interface for users. To have a mechanism that controls the totality of the requests reaching the site and to mask the service distribution among multiple servers, Web clusters provide a single virtual IP address that corresponds to the address of the front-end server(s). Independently of the mechanism that existing Web clusters use to routing the load, we refer to this entity as the *Web switch*. The Domain Name Server(s) for the Web site translates the site address (e.g., www.site.edu) into the IP address of the Web switch. In such a way, the Web switch acts as a centralized global scheduler that receives the totality of the requests and routes them among the servers of the cluster (see Figure .2.2).

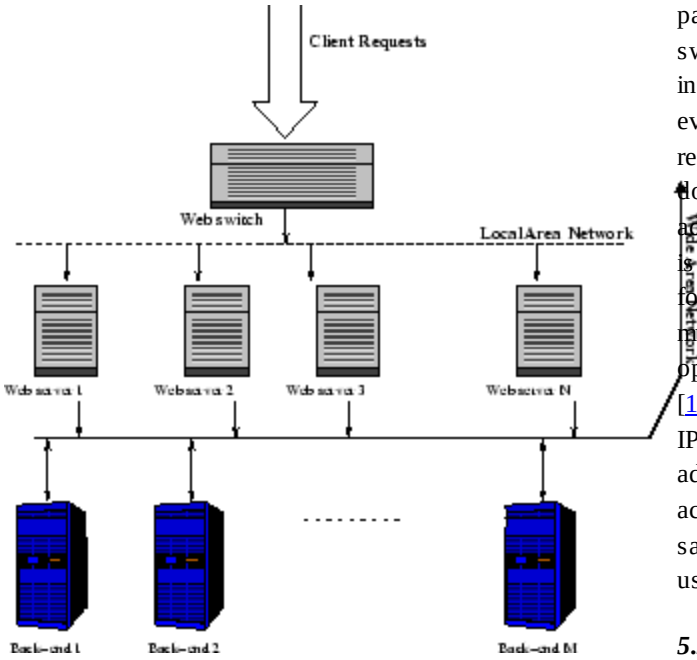


Figure 2.2: Web cluster architecture.

We consider a Web cluster consisting of homogeneous distributed servers that provide the same set of documents and services. The details about the operations of the Web cluster are described in Section 4.1. Various academic and commercial products confirm the increasing interest in these distributed Web architectures. In the *IBM TCP router* [17], all HTTP requests reach the Web switch that distributes them by modifying the destination IP address of each incoming packet: the Web switch replaces its IP address with the private address of the selected Web server. *Magic router* [2], *Distributed Packet Rewriting* [7] and Cisco *Local Director* [12] are other Web cluster architectures relying on a Web switch that receives the totality of client requests. In particular, Magic router is a mechanism of fast packet interposing where a user level process acting as a switchboard intercepts network packets and modifies them by changing addresses and checksum fields. Cisco Local Director rewrites the IP header information of each incoming packet according with a dynamic table of mapping between each session and the server to which it has been redirected. Unlike the TCP router that modifies only the client-to-server packets and lets the servers modify outgoing IP packets, Magic router and Local Director Web switches can be defined as *gateways* because they intervene even on server-to-client packets. An evolution of the TCP router architecture is represented by the IBM *Network Dispatcher* that does not require a modification of the packet addresses because packet forwarding to cluster nodes is done at the MAC address level [20]. A different forwarding approach to configure a Web system with multiple servers uses the if-configuration-alias option, which is available in most UNIX platforms [16]. This architecture publicizes the same secondary IP address of all Web servers as the IP single virtual address, namely *ONE-IP*, of the Web cluster. This is achieved by letting the servers of the cluster share the same IP address as their secondary address, which is used only for the request distribution service.

5.2 Web switches

A key component of any Web cluster is the Web switch that dispatches client requests among the servers. They can be broadly classified according to the OSI protocol stack layer at which they operate, so we have *layer-4* and *layer-7* Web switches [25]. Layer-4 Web switches work at TCP/IP level. 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. Layer-4 Web switch algorithms are *content information blind*, because they choose the target server when the client establishes the TCP/IP connection, before sending out the HTTP request. Global scheduling algorithms executable at the layer-4 Web switch range from static algorithms (say, random, round-robin) to dynamic algorithms that take into account either network client information, (say, client IP address, TCP port), or server state information (say, number of active connections, least loaded server) or even a combination of both information. Layer-7 Web switches can establish a complete TCP connection with the client and inspect

the HTTP request content prior to decide about dispatching. In such a way, they can deploy *content information aware* distribution, by letting the Web switch examine the HTTP request and then route it to the target server. The selection mechanism (usually referred to as *delayed binding*) can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies. In [5] there are many techniques to realize the dispatching granularity at the session level or at the single Web object request level. Scheduling algorithms deployed at layer-7 may use either client information (as session identifiers, file type, file size) or a combination of client and server state information. The potential advantages of layer-7 Web switches include the possibility to use specialized Web server nodes and partition the Web content among clusters of heterogeneous servers [28], and to achieve higher cache hit rates, for example, through affinity-based scheduling algorithms such as the LARD policy [24]. 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 blind and content aware request distribution have been proposed in [6,26]. These architecture solutions are out of the scope of this paper which is more focused on the dispatching algorithms for Web switches.

VI. WEB SWITCH ALGORITHMS

The Web switch may use various global scheduling policies to assign the load to the nodes of a Web cluster. Global scheduling methods were classified in several ways, depending on different criteria. The main alternatives are between load balancing *vs.* load sharing problems, centralized *vs.* distributed algorithms, static *vs.* dynamic policies. The Web cluster architecture with a single Web switch motivates the choice for *centralized* scheduling policies. If we consider that load balancing strives to equalize the server workload, while *load sharing* attempts to smooth out transient peak overload periods on some nodes, a Web switch should aim to share more than to balance cluster workload. Hence, the real alternative for layer-4 and layer-7 Web switches is the kind of system information they use to take assignment decisions. The main classes of policies are static and dynamic, these latter with several subclasses.

6.1 Static and dynamic global scheduling

Static policies do not consider any system state information. Typical examples are *Random* (RAN) and *Round-Robin* (RR) algorithms. RAN distributes the arrivals uniformly through the nodes. RR uses a circular list and a pointer to the last selected server to take dispatching decisions. *Dynamic policies* use some system state information while taking scheduling decisions. We consider the three classes of dynamic algorithms. *Server-aware algorithms* route requests on the basis of some server state information, such as load condition, latency time, availability or network utilization. *Client-aware algorithms* route requests on the basis of some client information. Layer-4 Web switches can use only some basic client network information, such as IP address and TCP port. Layer-7 Web switches can examine the entire HTTP request and take decisions on the basis of detailed information about the content of the client request. *Client- and server-aware algorithms* route requests on the basis of client and server state information. Actually, most of the existing client-aware algorithms belong to this class. Indeed, although the most important information is the client request, these policies combine it with some information about the server loads. The main goal is to avoid assignments to overloaded servers. 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, static algorithms are the fastest solution because they do not rely on the current state of the system at the time of decision making. For this reason, these algorithms can potentially make poor assignment decisions. Dynamic algorithms have the potential to outperform static algorithms by using

some state information to help dispatching decisions. On the other hand, dynamic algorithms require mechanisms that collect and analyze state information, thereby incurring potentially expensive overheads.

In this paper, we consider three widely used dispatching policies that are based on client and/or server information: *Weighted Round Robin* (WRR), *Locality Aware Request Distribution* (LARD) and *Static Partitioning*. WRR has resulted the layer-4 policy that guarantees best load sharing in most simulations and experiments from several research groups. On the other hand, we do not expect LARD to work well in a site providing heterogeneous services, but we have chosen it because we are not aware of other layer-7 dispatching algorithms proposed by the research community. *Static Partitioning* uses dedicated servers for specific services or multiple Web sites (co-location). This is the most representative example of a client-aware algorithm working at layer-7 in commercial Web switches [1,19].

WRR comes as a variation of the round robin policy. WRR associates to each server a dynamically evaluated weight that is proportional to the server load state [20]. Periodically (every T_{gat} seconds), the Web switch gathers this information from servers and computes the weights. WRR is actually a class of dynamic policies that uses some information about the system state. The first issue that needs to be addressed when we consider a server state aware policy is how to compute the load state information because it is not immediately available at the Web switch. The three main factors that affect the latency time are loads on CPU, disk and network resources. Typical load measures are the number of active processes on server, mean disk response time, and hit latency time, that is, the mean time spent by each request at the server. In particular, the load indexes we consider are the number of active processes at each server (*WRR_num* policy), and the mean service time for the requests (*WRR_time* policy). Additional information on WRR can be found in [20].

If we consider Web clusters of homogeneous servers, the main goal of the proposed policies is to augment disk cache hit rates, for example through the LARD policy [24] or other affinity-based scheduling algorithms [26,29]. The LARD policy [24] 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 [24] with the multiple hand-off mechanism defined in [5] 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 lowly loaded node, if it exists, or to the least loaded node. To this purpose, LARD defines two threshold parameters: T_{low} denoting the upper bound of a lowly loaded condition, and T_{high} denoting the lower bound of a highly loaded condition.

6.2 Client-aware policy

All previously proposed scheduling policies take static decisions independently of any state information (e.g., RAN and RR) or they take dynamic decisions on the basis of the state of the server nodes (e.g., WRR) that can be combined with client request information (e.g., LARD). We propose a *client-aware policy* (CAP) that takes into account some information associated to client requests as it can be gotten by a layer-7 Web switch. CAP, in its basic form, is a pure client-aware policy, however, it can be easily combined with some server state information. In this paper, we consider the pure CAP that does not gather any load information from servers. Pure client-aware policies have a possible great advantage over server-aware policies because server-aware algorithms often require expensive and hard to tuning mechanisms for monitoring and evaluating the load on each server, gathering the

results, and combining them to take scheduling decisions. In a highly dynamic system such as a Web cluster this state information becomes obsolete quickly. The key idea for CAP comes from the observation that dynamic policies such as WRR and LARD work fine in Web clusters that host traditional Web publishing services. In fact, most load balancing problems occur when the Web site hosts heterogeneous services that make an intensive use of different Web server's components. Moreover, almost all commercial layer-7 Web switches use client information for a static partitioning of the Web services among specialized servers [1,19]. The simulation experiments will confirm the intuition that a *Static Partitioning* policy, although useful from the system management point of view, achieves poor server utilization because resources that are not utilized cannot be shared among all clients. To motivate the CAP policy, let us classify Web services into four main categories.

Web publishing sites providing static information (e.g., HTML pages with some embedded objects) and dynamic services that do not intensively use server resources (e.g., result or product display requests). The content of dynamic requests is not known at the instant of a request, however, it is generated from database queries whose arguments are known before hand.

Web transaction sites providing dynamic content generated from (possibly complex) database queries built from user data provided by an HTML form. This is a disk bound service because it makes intensive use of disk resources.

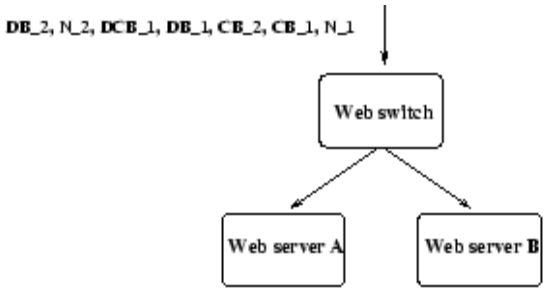
Web commerce sites providing static, dynamic and secure information. For security reasons, some dynamically generated content may need a secure channel that in most cases is provided by the SSL protocol. Cryptography makes intensive use of CPU resources. Hence, Web commerce services are disk and/or CPU bound.

Web multimedia sites providing streaming audio and video services. In this paper, we do not consider this type of application that often is implemented through specialized servers and network connections.

Although realistic, this classification is done for the purposes of our paper only and does not want to be a precise taxonomy for all Web services. 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. Any Web content provider can easily tune the CAP policy at its best. Starting from the above classification, we distinguish the Web requests into four classes: *static* and *lightly dynamic* Web publishing services (N); *disk bound* services (DB), for example, in Web transaction and Web commerce sites; *CPU bound* (CB) and *disk and CPU bound* (DCB) services, for example, in Web commerce sites. 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. We describe the CAP behavior through the following example.

We suppose that the server A has already received one request of type CB and one of type DCB; the server B has received one request of type N, and one of type DB. The sequence of successive requests to the Web cluster is shown in Figure 2. By using the CAP assignment, server A and B have a similar number of requests for each class of service, while this does not happen when using RR or LARD. For example, in the case of RR the server A receives four intensive requests that stress the CPU and/or disk, while server B receive only one CPU bound request. In the case of LARD, we suppose that the requests of type DB and CB are assigned to the server A and those of other types to the server B. This dispatching results that the server A receives two CPU bound and two disk bound requests, while the server B receives only one request of type DCB.

CAP does not require a hard tuning of parameters which is typical of most dynamic policies because the service classes are decided in advance, and the scheduling choice is determined statically once the URL has been classified.



Sequence Assignement

Algorithm	*Web server A*	*Web server B*
CAP	N_1, CB_2, DB_1	CB_1, DCB_1, N_2, DB_2
RR	N_1, CB_2, DCB_1, DB_2	CB_1, DB_1, N_2
LARD	CB_1, CB_2, DB_1, DB_2	N_1, DCB_1, N_2

	*	*					*	*		
Initial state	**0**	**0**	1	1	1	1	**0**	**0**
	N	DB	CB	DCB	N	DB	CB	DCB
CAP Final state	1	1	2	1	2	2	1	1
	N	DB	CB	DCB	N	DB	CB	DCB
RR Final state	1	1	2	2	2	2	1	**0**
	N	DB	CB	DCB	N	DB	CB	DCB
LARD Final state (DB, CB —> A N, DCB —> B)	**0**	2	3	1	3	1	**0**	1
	N	DB	CB	DCB	N	DB	CB	DCB

* denotes the next server in the assignment for each class of Web service

Figure 2.3: Example of behavior of CAP, RR and LARD dispatching policies.

VII. CONCLUSIONS & FUTURE WORK

A fundamental merit of the proposed algorithm is its ability to trace dead machines. Further it has capability to divide and distribute web request on the basis of processing power involved.

Our future work will focus on java implementation of this proposed algorithm and prove through simulation that this framework works well with heterogeneous web servers where incoming load is high and response is given within few seconds without any bottleneck.

One limitation left behind is suppose the main server controller fails then whole system will halt down. Because all requests are going through main server controller and if this server is unable to pass on the http request, framework will be shut down.

REFERENCES

[1] GUO Cheng Cheng YAN Pu Liu(School of Electronic Information, Wuhan University, Wuchang 430079)

[2] [2.1] Qi Zhang
Dept. of Comput. Sci., Coll. of William & Mary, Williamsburg, VA, USA
A. Riska ; W. Sun ; E. Smirni ; G. Ciardo

[2.2] Cardellini, Valeria; Colajanni, Michele; Yu, Philip S. **IEEE Internet Computing** 3.3 (May 1999): 28-39.

[3] Saeed Sharifian[a], Seyed A. Motamedi, Mohammad K. Akbari

[4] Gianfranco Ciardo, Alma Riska , Evgenia Smirni

[5] Zhang Lin, Xiao-Ping Li, Yuan Su

[6] Luciano Bertini[a] ,Julius C.B. Leite[a],Daniel Mossé

[7] Alteon WebSystems, Alteon 780 Series, in www.alteonwebsystems.com/products/

[8] E. Anderson, D. Patterson, E. Brewer, ``The Magicrouter, an application of fast

packet interposing", unpublished Tech. Rep., Computer Science Department, University of Berkeley, May 1996.

[9] Apache docs., in www.apache.org/docs/mod/

[10] M.F. Arlitt, C.L. Williamson, ``Internet Web servers: Workload characterization and performance implications", *IEEE/ACM Trans. on Networking*, vol. 5, no. 5, Oct. 1997, pp. 631-645.

[11] M. Aron, P. Druschel, W. Zwaenepoel, ``Efficient support for P-HTTP in cluster-based Web servers", Proc. USENIX 1999, Monterey, CA, June 1999.

[12] M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, ``Scalable content-aware request distribution in cluster-based network servers", Proc. USENIX 2000, San Diego, CA, June 2000.

[13] L. Aversa, A. Bestavros, ``Load balancing a cluster of Web servers using Distributed Packet Rewriting", *Proc. of IEEE IPCCC'2000*, Phoenix, AZ, February 2000.

[14] P. Barford, A. Bestavros, A. Bradley, M.E. Crovella, ``Changes in Web client access patterns: Characteristics and caching implications", World Wide Web, Jan. 1999.

[15] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1999.

[16] L. Anand, D. Ghose and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems," *Int',l J. Computers and Math. with Applications*, vol. 37, no. 8, pp. 57-85, Apr. 1999.

[17] J. Krallmann, U. Schwiegelshohn and R. Yahyapour, "On the Design and Evaluation of Job Scheduling Algorithms," *Proc. Fifth Workshop Job Scheduling Strategies for Parallel Processing*,pp. 17-42, 1999.

[18] G. Manimaran and C. Siva Ram Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 312-319, Mar. 1998.

[19] T Sorensen, P Mogensen, F Frederiksen, Extension of the ITU channel models for wideband (OFDM) systems, in *2005 IEEE 62nd Vehicular Technology Conference (VTC)*, vol. 1 (VTC-2005-Fall, Dallas, 2005), pp. 392–396

[20] P Kyösti, J Meinilä, L Hentilä, X Zhao, T Jämsä, M Narandzić, M Milojević, C Schneider, A Hong, J Ylitalo, V-M Holappa, M Alatossava, R Bultitude, Y de Jong, T Rautiainen, IST-4-027756 WINNER II D1.1.2. WINNER II channel models. Part II. Radio channel measurement and analysis results. v1.0. Tech. rep. WINNER II IST project 2007

[21] JM Ruiz-Avilés, S Luna-Ramírez, M Toril, F Ruiz, I de la Bandera, P Muñoz, R Barco, P Lázaro, V Buenestado, Design of a computationally efficient dynamic system-level simulator for enterprise LTE femtocell scenarios. Journal of Electrical and Computer Engineering **2012**, 14 (2012)