FLEX: Design and Management Strategy for Scalable Web Hosting Service

Ludmila Cherkasova
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-1999-64(R.1)
October, 1999

E-mail: {cherkasova}@hpl.hp.com

web hosting service, web server farm, web server cluster, virtual servers, DNS, load balancing, scalability, performance analysis, synthetic traces

We propose a new scalable solution for design and management of an efficient Web hosting service, called FLEX. This solution can be applied to a Web hosting service implemented on different architecture platforms, such as web server farms with replicated disk content, or web server clusters having access to a shared file system. FLEX is based on the "locality aware" balancing strategy which aims to avoid the unnecessary document replication to improve the overall performance of the system.

By monitoring the traffic to each customer's site and analyzing the combined traffic to a system in whole, FLEX proposes a balanced (logical) partitioning of the customers (web sites) by the number of nodes in the system.

The desirable routing can be done by submitting the correspondent configuration files to the DNS server, since each hosted web site has a unique domain name. FLEX can be easily implemented on top of the current infrastructure used by Web hosting service providers.

Using a simulation model and a synthetic trace generator, we compare the current solutions and FLEX over the range of different workloads. For generated traces, FLEX outperforms current solutions 2-5 times.

This paper is a continuation and development of the work started in [Ch99].

Contents

1 Introduction **3**

2 Shared Web Hosting: Typical Hardware Solutions **5**

3 Load Balancing Products and Solutions **7**

3.1 **DNS Based Approaches** 8

3.2 **IP/TCP/HTTP Redirection Based Approaches** 9

3.3 **Locality Aware Balancing Strategies** 13

4 Virtual Servers and Multiple Domains **14**

5 New Scalable Web Hosting Solution: FLEX **15**

6 Load Balancing Algorithm *flex-alpha* **17**

7 Algorithm *flex-alpha*: Analysis, Speed and Qualitative Results **20**

8 Synthetic Trace Generator **22**

9 Simulation Results with Synthetic Traces **23**

10 Conclusion and Future Research **25**

11 References **26**

A APPENDIX **28**

A.1 **Example 1: 50 Customers Balanced Across 10 Servers** 28

A.2 **Example 2: 100 Customers Balanced Across 10 Servers** 30

A.3 **Example 3: Synthetic Trace with 100 Customers Balanced Across 8 Servers** 32

A.4 **Example 4: Synthetic Trace with 100 Customers Balanced Across 16 Servers** 35

1 Introduction

Due to the explosive growth of the Web and an increasing demand on the servers, Web content hosting is an increasingly common practice. In Web content hosting, providers who have a large amount of resources (for example, bandwidth to the Internet, disks, processors, memory, etc.) offer to store and provide Web access to documents from institutions, companies and individuals who lack the resources, or the expertise to maintain a Web server, or are looking for a cost efficient, "no hassle" solution. The service is typically provided with a fee, though some servers do not charge fees for non-commercial accounts.

Demand for Web hosting and e-commerce services continues to grow at rapid pace. IDC [IDC98] forecast rapid growth for Web hosting over the next five years. In particular, businesses will invest heavily in Web hosting services following the resolution of the Y2K compliance issues.

IDC forecasts that Web hosting market will reach nearly $12 billion by 2002, i.e. Web hosting market will experience 30 times growth in 5 years.

Typical uses of a web site are wide and varied. The web is being used for communication, research, marketing, customer support, selling and collaborative working. Many businesses achieve payback within months owing to cost savings, attraction of new business, speed to market or better market intelligence.

More than two-thirds of all corporate web sites are now hosted (outsourced), accordingly to Forrester Research Inc.

Web hosting services are represented by the following market segments:

- *shared web hosting.* Shared servers host multiple sites from different companies;
- *dedicated web hosting*, which includes complex dedicated hosting and custom hosting. Dedicated servers belong to a single company and contain only its content.

One of the main benefits of shared web hosting is the *price* which can be *one-tenth* that of the dedicated platforms. Performance and security are the primary issues for shared web hosting. Shared web hosting forces companies to contend for server resources and LAN connections. Providing some performance guarantees will make shared web hosting a much more attractive solution.

Another interesting angle of web hosting is the proportion and the amount of servers used to support it.

For example, Digex Inc. (its market share is estimated by IDC to 2.9%) provides dedicated servers exclusively. Digex has a large Sun server web farm that supports Unix-based Web hosting, and it owns and operates the world's largest dedicated Web site management facility for Windows NT, with more than 500 Windows NT servers. Overall, Digex manages 900 dedicated servers for 650 customers.

We can speculate that with predicted growth for Web hosting services (8 times in next 3 years), the difficulty in management of thousands of dedicated servers might stimulate a larger growth of shared web hosting, especially if the performance and security issues are resolved.

A cluster (farm) of servers is used to increase the capacity and compute power of the solution.[1] Ideally, a cluster (farm) of N web servers should be N times more powerful than one web server.

However, to create a scalable solution one has to overcome a number of problems in a design: content management and load balancing.

Web server performance greatly depends on efficient RAM usage. A web server works faster when it pools pages from a cache in RAM. Moreover, its throughput is much higher too. We've measured web server throughput when it supplied files from the RAM (i.e. the files were already downloaded from disk and resided in the file buffer cache), comparing it against the web server throughput when it supplied files from the disk. Difference in throughput was more than 10 times.

One of the typical remedies to improve the web server performance is to increase RAM size and to configure a bigger file buffer cache. The significance of efficient RAM usage is difficult to underestimate.[2]

Load balancing (of either kind) for a cluster of web servers pursues the goal to equally distribute the load across the nodes. This solution interferes with another goal of efficient RAM usage for the cluster. The popular files tend to occupy RAM space in all the nodes. This redundant replication of "hot" content through the RAM of all the nodes leaves much less available RAM space for the rest of the content, leading to a worse overall system performance. Under such an approach, a cluster having N times bigger RAM (which is a combined RAM of N nodes) might effectively have almost the same RAM as one node, because of the replicated popular content through the RAMs in the cluster.

An orthogonal approach is to partition the content and in such a way to use RAM space more efficiently. However, static partitioning will inevitably lead to an inefficient, suboptimal and inflexible solution, since the changes in access rates as well as access patterns tend to vary dramatically over time, and static partitioning does not accommodate for this.

The observations above have led to a design of the new "locality aware" balancing strategies [LARD98] which aim to avoid the unnecessary document replication to improve the overall performance of the system.

In this paper, we introduce a new scalable, "locality aware" solution FLEX for design and management of an efficient Web hosting service. The solution can be applied to a shared web hosting service implemented on different architecture platforms such as web server farms, web

[1]Cluster (farm) of computers is a common way of improving availability too. However, in this paper, we mostly concentrate on scalability and performance issues.

[2]The case of our interest is when the overall file set is greater than the RAM of one node. If the file set completely fits to the RAM, any of existing load balancing strategies provides a good solution.

server clusters or multi-computer systems [HP-MCS]. FLEX can be easily implemented on top of the current infrastructure used by Web hosting service providers.

This paper is the second part of the work started in [Ch99]. In [Ch99], we analyzed the Web hosting market and its major segments, distinguished the main set of parameters which characterize the different web sites, outlined the FLEX solution idea and provided preliminary performance analysis of FLEX.

In this paper, we provide a more detailed survey and analysis of existing load balancing strategies and products. The FLEX solution and its "core" part – the load and content balancing algorithm *flex-alpha* – are described and analyzed in depth. We outline the synthetic trace generator, which has been used to generate a variety of different workload types to analyze the performance of FLEX. For generated traces, FLEX outperforms the current solutions 2-5 times.

ACNOWLEDGMENTS: The author would like to thank Denny Georg for motivation, sharing the ideas, encouragement and active support during all the stages of this work.

2 Shared Web Hosting: Typical Hardware Solutions

Web hosting is an infrastructure service that allows to design, integrate, operate and maintain all infrastructure components required to run web-based applications. It includes server farms, network access, data staging tools and security firewalls. Some business require an environment supporting business critical processes, which must offer 24x7 availability, enterprise-wide scalability, worldwide coverage and trusted security. Many other businesses have simpler requirements, and are operating web environments that do not have the same critical requirements.

Web server farms and clusters are used in a Web hosting infrastructure as a way to create scalable and highly available solutions.

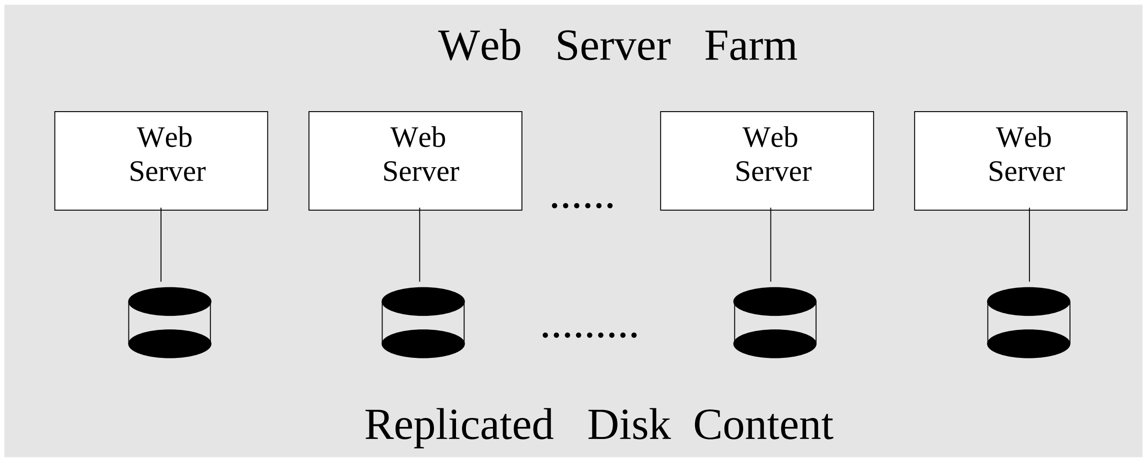

Figure 1: Web Server Farm with Replicated Disk Content.

One popular solution is a farm of web servers with replicated disk content shown in Figure 1.

This architecture has certain drawbacks:

- replicated disks are expensive, and
- replicated content requires content synchronization, i.e. whenever some changes to content data are introduced – they have to be propagated to all of the nodes.

Another popular solution is a clustered architecture, which consists of a group of nodes connected by a fast interconnection network, such as a switch. In a flat architecture, each node in a cluster has a local disk array attached to it.

As shown in Figure 2, the nodes in a cluster are divided into two *logical* types: front end (delivery, HTTP servers) and back end (storage, disks) nodes.

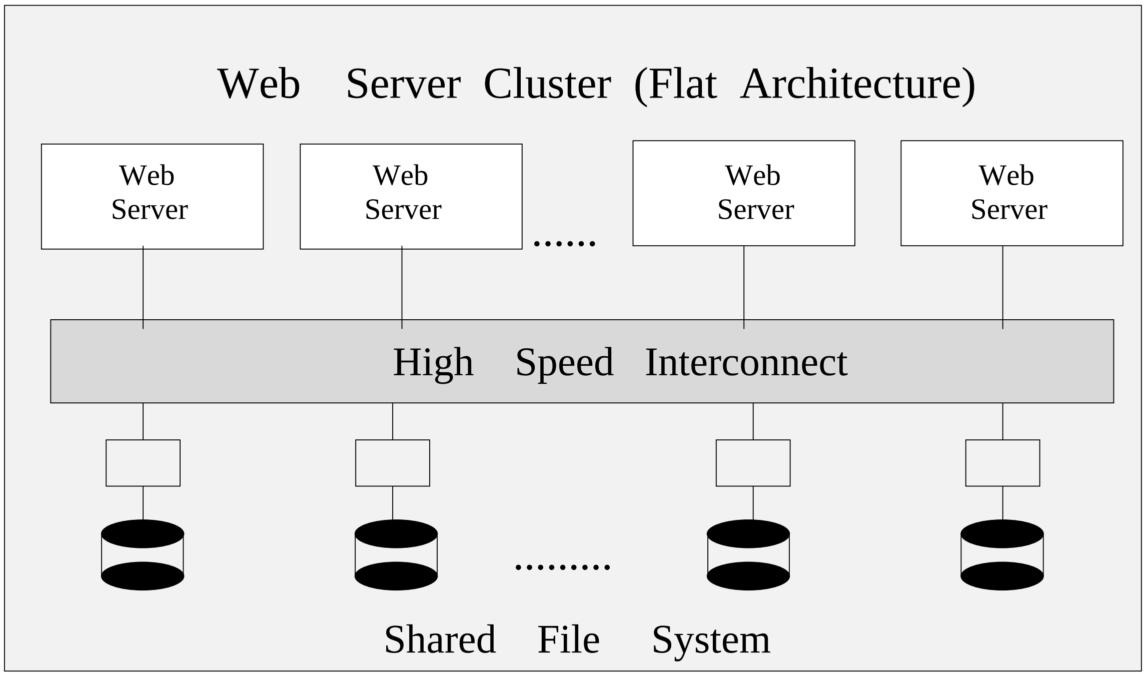

Figure 2: Web Server Cluster (Flat Architecture).

The (logical) front-end node gets the data from the back-end nodes using a shared file system. In a flat architecture, each physical node can serve as both the logical front-end and back-end, all nodes are identical, providing both delivery and storage functionality.

In a two-tiered architecture, shown in Figure 3, the logical front-end back-end nodes are mapped to different physical nodes of the cluster and are distinct. It assumes some underlying software layer (e.g., virtual shared disk) which makes the interconnection architecture transparent to the The NSCA prototype of the scalable HTTP server based on two-tier architecture is described and studied in [NSCA94, NSCA95, NSCA96].

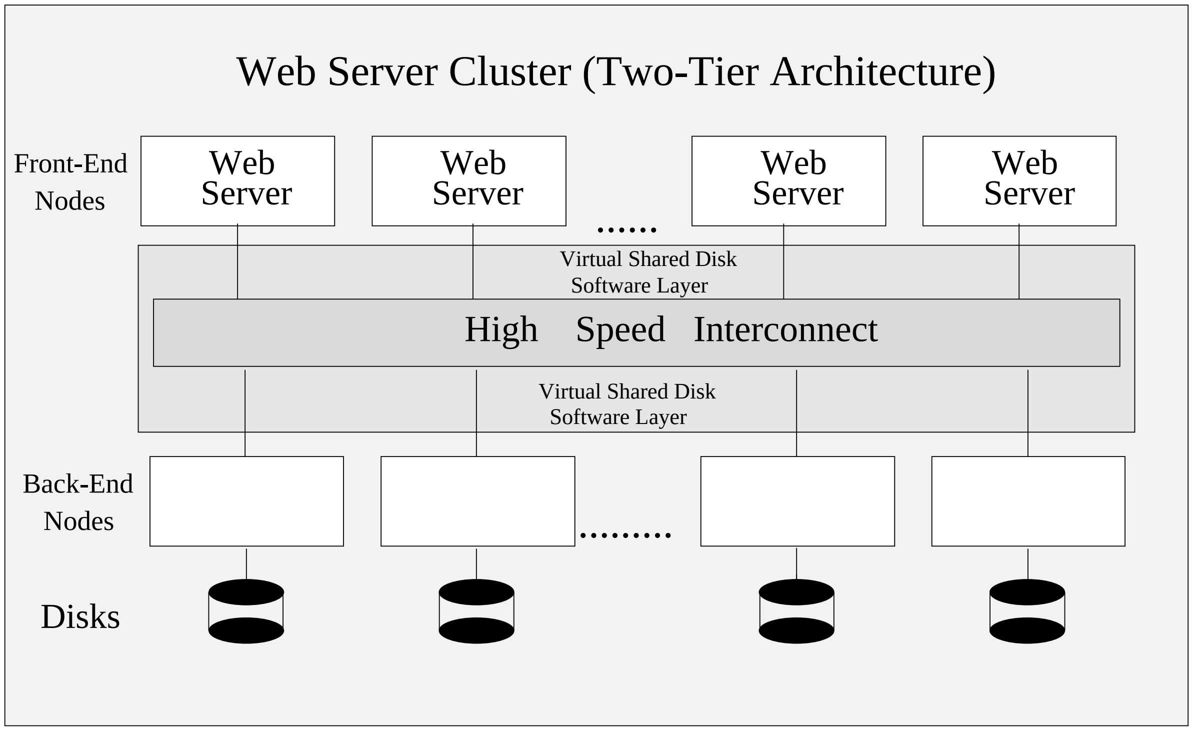

Figure 3: Web Server Cluster (Two Tier Server Architecture).

In all the solutions, each web server has the access to the whole web content. Therefore, any server can satisfy any client request.

What are the problems a service provider faces when trying to design a scalable and cost efficient solution using web server farms? One of the main problems in web server cluster (farm)[1] management is **content management and load balancing.**

3 Load Balancing Products and Solutions

The different products introduced on a market for load balancing can be partitioned in two major groups:

- DNS Based Approaches;
- IP/TCP/HTTP Redirection Based Approaches;
  - hardware load-balancers;
  - software load-balancers.

[1]We often use the terms of web server cluster and web server farm interchangeably, because the problems as well as the solutions are often very similar. Only in those cases when it matters, it is clearly specified.

3.1 DNS Based Approaches

Software load balancing on a cluster is a job traditionally assigned to a Domain Name System (DNS) server. **Round-Robin DNS** [RRDNS95] is built into the newer version of DNS. Round-Robin DNS distribute the access among the nodes in the cluster: for a name resolution it returns the IP address list (for example, list of nodes in a cluster which can serve this content, see Figure 4), placing the different address first in the list for each successive requests. Ideally, the different clients are mapped to different server nodes in a cluster.

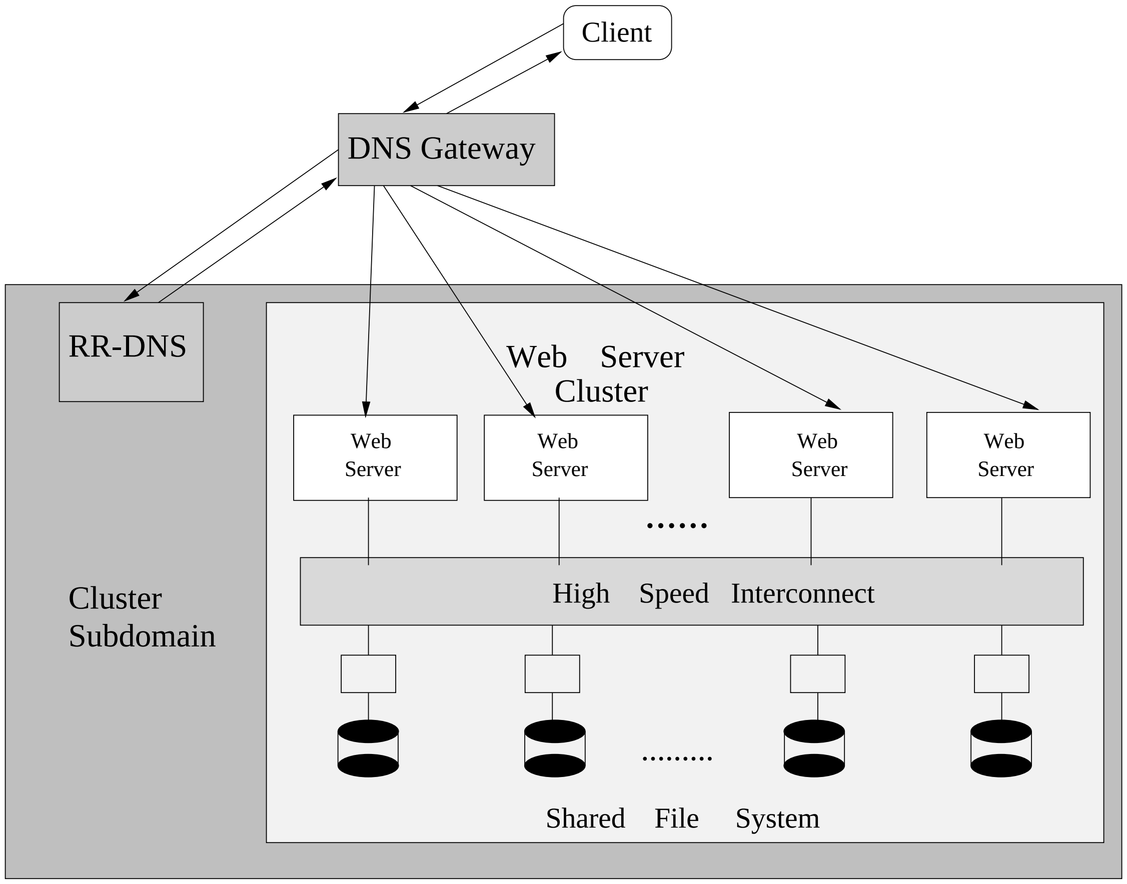

Figure 4: Web Server Cluster Balanced with Round-Robin DNS.

In most of the cases, Round-Robin DNS is widely used: it is easy to set up, it does provide reasonable load balancing and it is available as part of DNS which is already in use, i.e. there is no additional cost.

The Round-Robin schema has been critisized that it simply posts packets to the next server on the chain – without verifying whether it is already overloaded (partially, this problem occurs due to unequal capacity of servers). The second problem, is that if web server becomes unavailable, the DNS still keeps sending the requests to this server.

There are two products on a market that attempt to solve these problems: **Cisco Dis-**

tributedDirector and **HP Network Connection Policy Manager.**

Cisco DistributedDirector (DD) consists of DNS server software running on a Cisco 2500 or 4700 router hardware with some enhancements. In addition, a Cisco proprietary "DRP Protocol" is active on routers that attach the distributed web server farm to the rest of the network. A new subdomain is created for the web cluster. When a web request has to be routed to a web cluster, DD queries the DRP agents running on each of the routers for a number of metrics, such as distance from that router to the client, administrative weight of the server, server availability, etc. On the basis of the above metrics (one or more can be administratively configured) the DD selects the best server, and returns that IP address to the client.

HP Network Connection Policy Manager (NCPM) may be also used for load balancing on web cluster (although it was not explicitly designed for this purpose). NCPM is a software add-on that is co-resident with the BIND DNS server running on HP-UX. In addition, a host agent must be running on each web server. The host agent periodically (period is user configurable) sends a status message to the NCPM Manager with values for a few metrics such as:

- CPU Utilization
- Free memory
- Free swap space
- Number of TCP connections
- Number of running processes
- one user-definable metric.

At most 2 metrics can be used to select possible candidates in a cluster, and one metric is used to sort selected servers in order from best to worst. This list is then returned to the client, just as in simple Round Robin DNS.

The user defined metric can be any script written by the user, as long as it returns a numeric value that can be processed by NCPM.

3.2 IP/TCP/HTTP Redirection Based Approaches

The market now offers several hardware/software load-balancer solutions that can distribute incoming stream of requests among a group of Web servers.

One group of load balancers are often called as **load-balancing servers**: they intelligently distribute incoming requests across the multiple web servers. These products can decide where to send incoming request, taking into account the processing capacity of attached servers, monitoring the responses in real time and shifting the load onto servers that can best handle the traffic.

There are at least nine representatives of load-balancing servers:

- six are standalone units (hardware load balancers);
- three are software packages that can be loaded onto network servers.

Four of the six hardware load balancers on the market are built around Intel pentium processors:

- LocalDirector from Cisco Systems [Cisco],
- Fox Box from Flying Fox [FlyingFox],
- BigIP from F5 Labs [F5Labs], and
- Load Manager 1000 from Hydraweb Technologies Inc. [HydraWEB].

Another two load balancers employ a RISC chip:

- Web Server Director from RND Networks Inc. [RND] and
- ACEdirector from Alteon [Alteon].

All these boxes except Cisco's and RND's run under Unix. Cisco's LocalDirector runs a derivative of the vendor's IOS software; RND's Web Server Director also runs under a proprietary program.

Three software load-balancing servers also are available:

- ClusterCATS from Bright Tiger Technologies [BrightTiger],
- SecureWay Network Dispatcher from IBM [IBM-SWND], and
- Central Dispatch from Resonate Inc [Resonate].

These products are loaded onto Unix or Windows NT servers.

Hardware and software load balancers can perform double duty. Load-balancing boxes are basically IP routers and can offload their corporate counterparts. Software balancers also can function as Web servers.

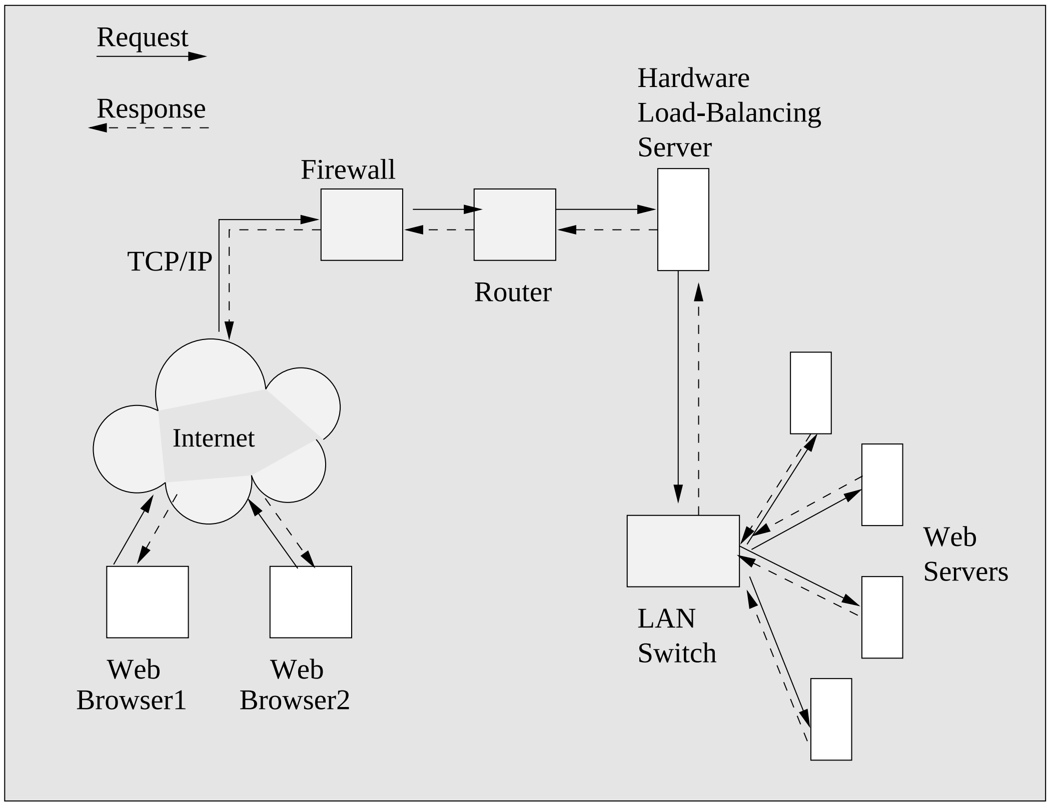

Figure 5: Web Server Farm with Hardware Load-Balancing Server.

Load-balancing servers are typically positioned between a router (connected to the Internet) and a LAN switch which fans traffic to the Web servers. Typical configuration is shown in Figure 5.

In essence, they intercept incoming web requests and determine which web server should get each one. Making that decision is the job of the proprietary algorithms implemented in these products. This code takes into account the number of servers available, the resources (CPU speed and memory) of each, and how many active TCP sessions are being serviced. The algorithms also monitor how efficiently each web server is processing requests by time-stamping a packet and seeing how long it takes to reach the server and be returned. In theory, faster servers are sent more requests than their slower counterparts. The balancing methods across different load-balancing servers vary, but in general, the idea is to forward the request to the least loaded server in a cluster.

Load-balancing server acts as a fast regulating valve between the Internet and the pool of servers. The load balancer uses a virtual IP address to communicate with the router, masking the IP addresses of the individual servers. Only the virtual address is advertised to the Internet community, so the load balancer also acts as a safety net.

The IP addresses of the individual servers are never sent back to the Web browser. If they were, the browser would try to establish a session with a specific Web server, rather than with the load-balancing server. This would defeat the entire purpose of deploying a load-balancing server to distribute requests. Both inbound requests and outbound responses must pass through the balancing server.

The software load balancers take a different tack, handing off the TCP session once a request has been passed along to a particular server. In this case, the server responds directly to the browser (see Figure 6). Vendors claim that this improves performance: responses don't have to be rerouted through the balancing server, and there's no additional delay while an internal IP address of the server is retranslated into an advertised IP address of the load balancers.

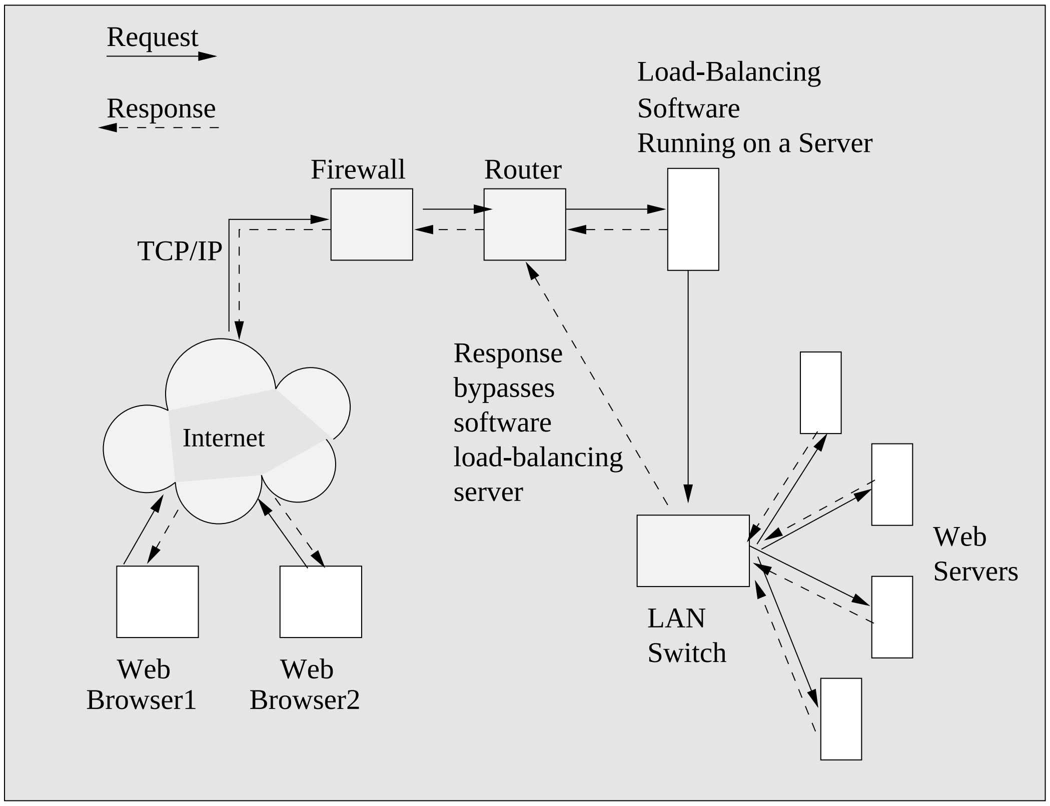
Request
Response
Load-Balancing Software Running on a Server
Firewall
Router
TCP/IP
Internet
Response bypasses software load-balancing server
Web Servers
LAN Switch
Web Browser1
Web Browser2

Figure 6: Web Server Farm with Load-Balancing Software Running on a Server.

Actually, that translation is handled by the Web server itself. Software load balancers are sold with agents that must be deployed on the Web server. It's up to the agent to put the right IP address on a packet before it's shipped back to a browser. If a browser makes another request, however, that's shunted through the load-balancing server.

Hardware switches mentioned above are expensive (Cisco Local Director cost around $32,000). They might significantly increase a solution cost. All traffic to the content is directed through the switch. If only one switch is used then it introduces a single point of failure. Minimal configuration of two switches increases the solution cost even further. Clearly, scalability might be a problem, since a switch could become a bottleneck. In fact, similar concerns are applied to software load balancers as well.

More information on load balancing products and solutions can be found in [Bruno97, Roberts98].

To complete the survey, the latest, different approach which vendors of network devices are undertaken, is worth mentioning. Now some vendors of network devices propose to delegate load balancing to linchpins like switches and firewalls. The advantages seem hard to ignore. For one thing, it saves money. For another, it brings better scalability and performance: balancing is performed in speedy hardware instead of slow software, and with functions combined the number of network devices through which traffic passes is reduced.

In fact, of the six vendors now entering new territory, switch makers represent the majority. Alteon Networks Inc. (San Jose, Calif.), Arrowpoint Communications Inc. (Westford, Mass.), Foundry Networks Inc. (Sunnyvale, Calif.), and Holontech Corp. (San Jose) all are building load balancing into their LAN switches. Check Point Software Technologies Inc. (Redwood City, Calif.), sells an optional software module that lets its Firewall-1 firewall distribute loads across servers. Finally, there's Allot Communications Inc. (Los Gatos, Calif.), which has built load balancing into its AC200 and AC300 traffic tuner products.

Most of these devices perform NAT (network address translation), so that a single virtual IP address can be used by the switch, firewall, or traffic tuner to represent multiple servers. The exception is Holontech's Hyperflow SP800, which relies on clustering technology from the operating system vendors to allow a server farm to share a virtual IP address.

3.3 Locality Aware Balancing Strategies

Load balancing for a cluster (farm) of web servers pursues the goal to equally distribute the load across the nodes. This solution may prevent the efficient usage of RAM in the cluster (farm).[1] There is a redundant replication of "hot", popular content through the RAM of all the nodes which leaves much less of available RAM space for the rest of the content. This may significantly decrease overall system performance. This observation have led to a design of the new "locality aware" balancing strategies [LARD98]. The elements of the "content aware" strategies can be found in last products of Alteon Networks Inc. (San Jose, Calif.) and Arrowpoint Communications Inc. (Westford, Mass.

A new locality-aware request distribution strategy (LARD) is proposed for cluster-based network servers in [LARD98]. The cluster nodes are partitioned into two sets: front ends and back ends. Front ends act as the smart routers or switches: their functionality is similar to load-balancing software servers described above. Front end nodes implement LARD to route the incoming requests to the appropriate node in a cluster. LARD takes into account both a document locality and the current load. Authors show that on workloads with working sets that do not fit in a single server nodes RAM, the proposed strategy allows to improve

[1]We consider only workloads with working set that does not fit in a single server nodes RAM.

throughput by a factor of two to four for 16 nodes cluster.

In this paper, we introduce a new scalable solution FLEX for design and management of an efficient Web hosting service. FLEX motivation is similar to the "locality aware" balancing strategies discussed above: we would like to avoid the unnecessary document replication to improve the overall performance of the system.

However, we achieve this goal via logical partition of the content on a different *granularity level.* Since the original goal is to design a scalable web hosting service, we have a number of customers and their sites as a starting point. Each of these sites might have different traffic patterns in terms of both the number and types of files accessed and the average access rates. By monitoring the traffic to each site and analyzing the combined traffic to a system in whole, FLEX proposes a balanced partitioning of the customers (web sites) by the number of nodes in the system.

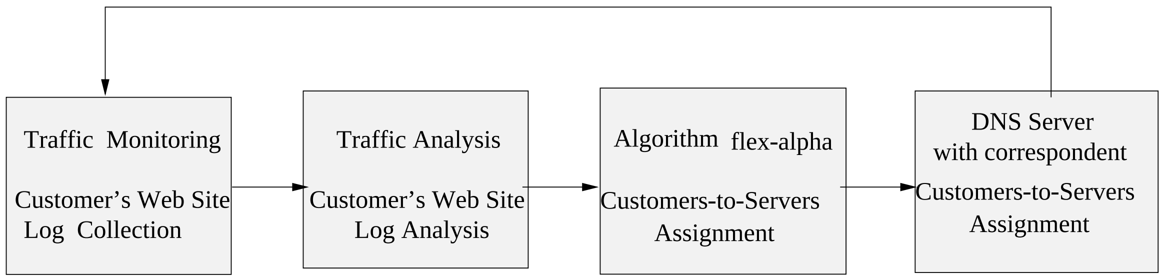
Traffic Monitoring
Customer's Web Site Log Collection
Traffic Analysis
Customer's Web Site Log Analysis
Algorithm flex-alpha
Customers-to-Servers Assignment
DNS Server with correspondent
Customers-to-Servers Assignment

Figure 7: FLEX Strategy: Logic Outline.

FLEX can be easily implemented on top of the current infrastructure used by Web hosting service providers. The desirable routing can be done by submitting the correspondent configuration files to the DNS server, since each hosted web site has a unique domain name. The DNS server is going to route the incoming requests to a correspondent node in the system (accordingly to partition provided by FLEX). The logic of the FLEX strategy is shown in Figure 7.

4 Virtual Servers and Multiple Domains

Although the resent survey revealed over 1 milion web servers on the Internet, the number of web site exceeds this number several times. The illusion of more web sites existing than actual web servers is created through the use of *virtual servers (hosts).*

Web hosting service is based on this technique. Web hosting service uses the possibility to create a set of virtual servers on the same server. There are different alternatives how it can be done.

Unix web servers (Netscape and Apache) have the most flexibility to address the web hosting problem. Multiple host (domain) names can be easily assigned to a single IP address. It allows to create an illusion that each host has its own web server, when in reality multiple "logical" hosts share one physical host.

There is also a possibility to create a separate IP address per each host. It could be used as well in Web hosting solution when the number of hosted sites is rather limited. It is less scalable when a web hosting service is dealing with a large number of relatively small sites.

For a while, in NT world, the second alternative was the only alternative. WWW Publishing Service can be configured to answer requests for more than one single domain name. To accomplish this, one should requests the IP addresses for the primary server and for each additional virtual server. Finally, these additional IP addresses have to be included in TCP/IP protocol configuration and DNS (or WINS) configuration files in order to resolve the IP addresses to the correspondent domain names. As noticed above, this solution is a good solution when the number of hosted sites is rather limited. It does not scale.

Last version of NT web server (IIS-4) introduces a new feature which allows to create multiple host names which can be assigned to the same IP address. This feature makes NT web server world to look similar to the Unix one.

One way to implement a Web Hosting Service will be to use a web server farm (or web server cluster) which has access to the whole content (whether it is achieved with replicated content or shared distributed file system). For example, there are total of 100 different sites (customers) which would like to publish their information on WWW. Each physical web server creates a virtual server per customer site, and announces prescribed IP addresses to correspondent domain names via DNS server configuration files.

For load balancing in such a cluster, Round Robin DNS is a typical solution. The disadvantages of such an approach were discussed in Subsection 3.3.

The other simple way to approach the load balancing in such a cluster, is to statically partition and assign the customers to the servers. For example, 100 customers could be partitioned as 10 customers per server in the configuration of 10 web servers. However, any such static partition can not take into account changing traffic patterns as well as nature of changes in the content of the sites. So, it can not adjust the partition to accommodate and provision for the traffic and sites dynamics.

Next Section 5 proposes a "logical partition" and assignment of customers to servers in a special way via the DNS server as a "router".

5 New Scalable Web Hosting Solution: FLEX

A new scalable solution, called FLEX, for shared Web Hosting consists in the following. By monitoring the access patterns and access rates to the customers content, the overall content can be **logically partitioned** in a number of "equally balanced" groups by the number of cluster (farm) nodes. Each customer group is serviced by some prescribed node in a cluster (farm).

For example, there is a total of C customers hosted on a cluster (farm) of N web servers. For each customer c, a "customer profile" CP_c is built. A customer profile CP_c consists of two following basic characteristics:

- AR_c - the access rates to a customer's content, i.e. *bytes/sec* requested of this customer content.
- WS_c - the total size of the most often requested files, so-called "working set".

The next step is to partition all the customers in N "equally balanced" groups: $C_1, ..., C_N$ in such a way, that cumulative access rates and cumulative "working sets" in each of those C_i groups are approximately the same. We designed a special algorithm, called *flex-alpha*, which does it. The next Section 6 describes the implementation details of *flex-alpha*.

The final step is to prescribe a web server N_i from a cluster (farm) to each group C_i.

REMARK1: The variation of the algorithm can be used for additional load (rate) balancing. For example, one site has a high bursty traffic. To smooth the high access rates this site, the site can be assigned to be served by two or more servers (depending on the balancing goal and a threshold for desirable rate).

REMARK2: This algorithm can be used to provide a desirable degree of high availability, additionally to load balancing. For example, the algorithm can prescribe minimum 2 (or 3, if desired) nodes per site to increase the site availability in case of the nodes failure. Note, that the load balancing is easier in such configuration since the rate per customer site decreases correspondingly to the number of nodes prescribed.

The FLEX solution is supported by providing the correspondent information to a DNS server via configuration files. A resolution of the customer domain name is going to be a correspondent IP address on the prescribed node (nodes) in the cluster (farm). This solution is flexible and easy to manage. Tuning can be done on a daily, weekly or even hourly (if necessary) basis. Once the server logs analysis shows enough changes in the average traffic rates and patterns and finds a new, better partitioning of the customers for this cluster (farm), then new DNS configuration files are generated. Once a DNS server has updated its configuration tables,[1] new requests are routed accordingly to the new configuration files, which leads to more efficient traffic balancing on a cluster (farm).

Such a self-monitoring solution allows to observe changing users' access behaviour and to predict future scaling trends, and plan for it. This solution could also be used to provision some special advertisement or promotion campaigns when one could expect very high traffic

[1]The entries from the old configuration tables can be cached by some servers and used for request routing without going to DNS server. However, the cached entries are valid for a limited time only dictated by TTL (*time to live).* Once TTL is expired, the DNS server is requested for updated information. During the TTL interval, both types of routing: old and a new one, can exist. This does not lead to any problems since any server has the access to the whole content and can satisfy any request.

rates for a certain content during some period of time. In those cases, for example, "hot" content can be prescribed to access via all the nodes in a cluster (farm).

6 Load Balancing Algorithm *flex-alpha*

We designed a special algorithm, called *flex-alpha*, which partitions the overall content in a number of "equally balanced" groups by the number of cluster (farm) nodes. Each group of customers is serviced by some prescribed node in a cluster (farm).

This Section describes the implementation details of *flex-alpha*. We use the following notations:

- $NumCustomers$ – a number of customers hosted on a web cluster (farm).
- $NumServers$ – a number of nodes (servers) in a web cluster (farm).
- $CustomerWS[i]$ – an array which provides the total size of the most frequently requested files of the i-th customer content, so-called "a working set" for the i-th customer. Without a loss of generality, we assume that the customers are ordered by the working set, i.e. the array CustomerWS[i] is ordered.
- $CustomerRate[i]$ – an array which provides the access rates to the i-th customer content, i.e. *bytes/sec* requested of this the i-th customer content.

Our goal is to assign the customers to the servers in such a way that balances the size of the working set and the access rates for each server. We will call such an assignment as *partition.*

At first, we compute the ideal balance *Even* we aim to achieve per server with respect to balancing the customers working set across the servers.

$$WorkingSetTotal = \sum_{i=1}^{NumCustomers} CustomerWS[i]$$

$$Even = \frac{WorkingSetTotal}{NumServers}$$

Second, we are going to normalize the access rates. It is done in two steps. We compute the sum of all customers rates $RatesTotal$ (in bytes/sec):

$$RatesTotal = \sum_{i=1}^{NumCustomers} CustomerRate[i]$$

The $RatesTotal$ represents $100\% * NumServers$. After that, the normalized access rate (in %) for each i-th customer is computed in the following way:

$$CustomerRate[i] = \frac{100\% * NumServers * CustomerRate[i]}{RatesTotal}$$

Now, the overall goal can be rephrased in the following way: we aim to partition all the customers in *NumServers* "equally balanced" groups: $C_1, ..., C_N$ in such a way, that

- cumulative "working sets" in each of those C_i groups is close to *Even*[1] and
- cumulative access rates in each of those C_i groups are around 100%.

The pseudo-code[2] of the core fragment of the algorithm *flex-alpha* is shown below in Figure 8. In this pseudo-code, we use additional notations:

- *CustomersLeftList* – the ordered list of customers which are not yet assigned to the servers. In the beginning, the *CustomersLeftList* is the same as the original ordered list of customers *CustomersList*;
- *ServerAssignedCustomers[i]* – the list of customers which are assigned to the i-th server;
- *ServerWS[i]* – the cumulative "working set" of the customers currently assigned to the i-th server;
- *ServerRate[i]* – the cumulative "access rate" of the customers currently assigned to the i-th server.
- $abs\_dif(x, y)$ – the absolute difference between x and y, i.e. $(x - y)$ or $(y - x)$, whatever is positive.

The assignment of the customers to all the servers except the last one is done accordingly to the pseudo-code in Figure 8. Fragment of the algorithm shown in Figure 8 above is applied in a cycle to the the first $NumServers - 1$ servers.

All the customers which are left in *CustomersLeftList* are assigned to the last server.

This completes one iteration of the algorithm, resulting in the assignment of all the customers to the servers in balanced groups. Typically, this algorithm generates a very good balancing with respect to the cumulative "working sets" of the customers assigned to the servers, because of the construction.

The second goal is to balance the cumulative access rates per server.

[1]Working sets can be normalized too. In Appendix, the Examples 3,4 are derived from the synthetic traces and have normalized working sets. For the normalized working sets, *Even*=100.

[2]We only describe the basic case. In exceptional situations such as when the size of working sets of some sites are larger than *Even*, and balanced solution does not exist, the algorithm provides correspondent warnings and produces the "best effort" solution.

```
/*
 *  we assign customers to the i-th server from the CustomersLeftList
 *  using random function until the addition of the chosen customer
 *  content  does not  exceed the ideal content limit per server  Even.
 */
    customer = random(CustomersLeftList);
    if (ServerWS[i] + CustomerWS[customer]) <= Even) {
        append(ServerAssignedCustomers[i], customer);
        remove(CustomersLeftList, customer);
        ServerWS[i] = ServerWS[i] + CustomerWS[customer];
        ServerRate[i] =  ServerRate[i] +  CustomerRate[customer];
    }
    else {
/*
 *  if the addition of the chosen customer content exceeds
 *  the ideal content limit per server Even
 *  we try to find such a  last_customer from the CustomersLeftList
 *  which results in a minimum deviation from the SpaceLeft
 *  on this server.
 */
      SpaceLeft = Even - ServerWS[i];
      find last_customer with min(abs_dif(SpaceLeft - CustomersWS[last_customer]));
      append(ServerAssignedCustomers[i], last_customer);
      remove(CustomersLeftList, last_customer);
      ServerWS[i] = ServerWS[i] + CustomersWS[last_customer];
      ServerRate[i] = ServerRate[i] +  CustomersRate[last_customer];
    }
    if (ServerWS[i]) > Even) {
/*  small optimization at the end: returning the customers with smallest
 *  working sets back to the CustomersLeftList until the deviation
 *  between the server working set ServerWS[i] and the ideal content
 *  per server Even is minimal.
 */
     if (abs_dif(Even - (ServerWS[i] - CustomersWS[redundant_customer])) <
         abs_dif(Even - (ServerWS[i])) {
         append(CustomersLeftList, redundant_customer);
         remove(ServerAssignedCustomers[i], redundant_customer);
         ServerWS[i] = ServerWS[i] + CustomersWS[redundant_customer];
         ServerRate[i] = ServerRate[i] + CustomersRate[redundant_customer];
     }
```

Figure 8: Pseudo-code of the core fragment of the algorithm *flex-alpha*.

For this purpose, for each partition P generated by the algorithm, the rate deviation of P is computed:

$$RateDeviation(P) = \sum_{i=1}^{NumServers} abs\_dif(100, ServerRate[i])$$

We define partition P_1 is *better rate-balanced* than partition P_2 iff

$$RateDeviation(P_1) < RateDeviation(P_2).$$

The algorithm is programmed to generate partition accordingly to the rules shown above. The number of iterations is prescribed by the input parameter *Times.* On each step, algorithm keeps a generated partition only if it is *better rate-balanced* than the previously best found partition.

Typically, the algorithm generates a very good balancing partition with respect to both: cumulative "working sets" and cumulative access rates of the customers assigned to the servers in 10,000 - 100,000 iterations. Next Section gives few examples of the algorithm results and the algorithm performance.

7 Algorithm *flex-alpha*: Analysis, Speed and Qualitative Results

In Appendix, we show few examples of different customers sets and the best partition generated by the *flex-alpha* algorithm on different iterations.

Here, we give only the summary of the algorithm results and the algorithm speed.

Speed of the algorithm linearly depends on the number of iterations the algorithm is prescribed to perform.

Speed of each iteration, in its turn, depends on the number of customers and servers in the input set. The higher the number of customers and servers - the longer it takes to finish the iteration.

But in general – the algorithm is fast: for a set of 50 customers to be balanced on 10 servers (see the Example 1 in Appendix A.1) the algorithm spent:

- to accomplish 10,000 iterations – 5.5 sec;
- to accomplish 100,000 iterations – 54.7 sec;
- to accomplish 1,000,000 iterations – 551.1 sec;

The new best partitions were found during the following iterations shown in Figure 9. We show the percentage of deviation of the generated partition from the optimal one. Here and further, by the *optimal partition* we mean a partition with *Even* size for "working set" per server, and 100% (normalized) access rates per server. Clearly, for some customers sets, this "optimal" partition may be not achievable. However, it is a good metric to regard to when evaluating the algorithm results.

Iteration	WS_Deviation(%)	Rate_Deviation(%)
0	1.39394	14.4923
1	0.40404	12.1055
5	0.484848	9.50175
14	0.323232	9.2801
24	0.343434	8.86048
48	0.808081	8.03578
83	0.464646	8.02679
103	0.585859	5.50325
2,168	0.707071	5.24268
2,728	0.808081	5.04131
5,774	0.484848	4.78754
5,937	0.525253	4.19492
33,608	0.484848	3.61023
80,664	0.484848	3.56745
145,640	0.989899	3.55944
165,040	0.444444	2.43375
236,698	0.444444	2.03177

Figure 9: Summary Results for a Set of 50 Customers Balanced Across 10 Servers.

Figure 9 shows that during the first 100,000 iterations the algorithm has found several *better rate-balanced* partitions. After that, the next improvements took significantly longer time. The quality of the results produced by the algorithm during the first 100,000 iterations is very high: it insignificantly deviates from the *optimal partition*.

Now, we consider the algorithm results for a set of 100 customers to be balanced on 10 servers (we used a similar distribution as for 50 customers in the previous example).

For a set of 100 customers to be balanced on 10 servers(see the Example 2 in Appendix A.2) the algorithm spent:

- to accomplish 10,000 iterations – 13.6 sec;
- to accomplish 100,000 iterations – 135.7 sec;

- to accomplish 1,000,000 iterations – 1356.4 sec;

The new best partitions were found during the following iterations shown in Figure 10.

Iteration	WS_Deviation(%)	Rate_Deviation(%)
0	0.184894	8.97584
2	0.114084	8.68343
3	0.228167	8.16569
7	0.236035	5.40416
49	0.267507	5.32785
92	0.0944142	4.792
110	0.0983484	4.77531
663	0.133753	4.48847
858	0.204564	4.04967
1,272	0.153423	3.70506
2,752	0.0747447	3.52089
3,308	0.0983484	3.29225
5,462	0.165225	3.00555
8,020	0.184894	2.96249
9,768	0.29111	2.85959
21,771	0.118018	2.03705
148,067	0.0865465	1.74509

Figure 10: Summary Results for a Set of 100 Customers Balanced Across 10 Servers.

First of all, the results are better than for the set of 50 customers. The number of servers in the example did not change. Having a bigger set of customers, allows richer combination choices, and, typically, better qualitative results. The other interesting observation is that the best results were received on the 148,067 iteration, and no improvement was found during the next 851,933 iterations.

8 Synthetic Trace Generator

We developed a synthetic trace generator to evaluate the performance and efficiency of proposed load balancing strategy FLEX.

There is a set of basic parameters which defines the traffic pattern, file distribution, and "web site" profiles in generated synthetic trace:

1. *NumCustomers* - number of web sites or (in other words) customers sharing the cluster;

2. *NumWebServers* - number of web servers in the targeted configuration. This parameter is used to define a number of directories used in the trace. We use a simple scaling rule: trace targeted to run on N nodes configuration has N times greater number of directories (and files) than single node configuration;

3. *OPS* - a single node capacity similar to SpecWeb96 benchmark. This parameter is only used to define the number of directories and file mix on a single server. Accordingly to SpecWeb96, each directory has 36 files from 4 classes: 0 class files are 100bytes-900bytes (with access rate of 35%), 1 class files are 1Kbytes-9Kbytes (with access rate of 50%), 2 class files are 10Kbytes-90Kbytes (with access rate of 14%), 3 class files are 100Kbytes-900Kbytes (with access rate of 1%).

4. *MaxCustSize* - a desirable maximum size of normalized working set per web site (per customer).

$$WorkingSetTotal = \sum_{i=1}^{NumCustomers} CustomerWS[i]$$

   The $WorkingSetTotal$ represents $100\% * NumServers$.
   The normalized working set $NormCustomerWS[i]$ is computed in the following way:

$$NormCustomerWS[i] = \frac{100\% * NumServers * CustomerWS[i]}{WorkingSetTotal}$$

   The *MaxCustSize* is used as an additional constraint:

$$NormCustomerWS[i] \leq MaxCustSize.$$

5. *RateBurstiness* - a range for a number of consequent requests to the same customer web site. For each customer web site, we generate its *RateBurstiness* as $CustRate[i] = random(1, RateBurstiness)$. In the trace, the requests to the particular customer web site, say i, are represented with a length $CustRate[i]$ requests sequence to different files in customer i content.

6. *TraceLength* - a length of the trace.

REMARK: Items 2 and 3 can be specified differently to define different file distribution mixture (currently, it is SpecWeb96 like) and different scaling rules.

9 Simulation Results with Synthetic Traces

In order to estimate the performance benefits of a new solution as well as to illustrate the pitfalls of the current solutions, the high level simulation model of web cluster (farm) has been built using C++Sim [Schwetman95].

Synthetic traces allow to create different traffic patterns, different files distributions, and different "web site" profiles. This variety is very useful to evaluate the designed FLEX strategy over wide variety of possible workloads. Evaluation of the strategy on a base of the real web sites is a next step in our research. But for now, we "probed" our strategy over different synthetic traces.

The first trace was generated for 100 customers and 8 web servers, each with capacity of 1000 OPS, with *MaxCustSize = 30%*, and *RateBurstiness= 30*. The length of the trace was 20,000,000 requests.

The "customer profile" analysis and the results of FLEX balancing algorithm for this trace are shown in Appendix A.3.

The simulation results for this trace are shown in Figure 11.

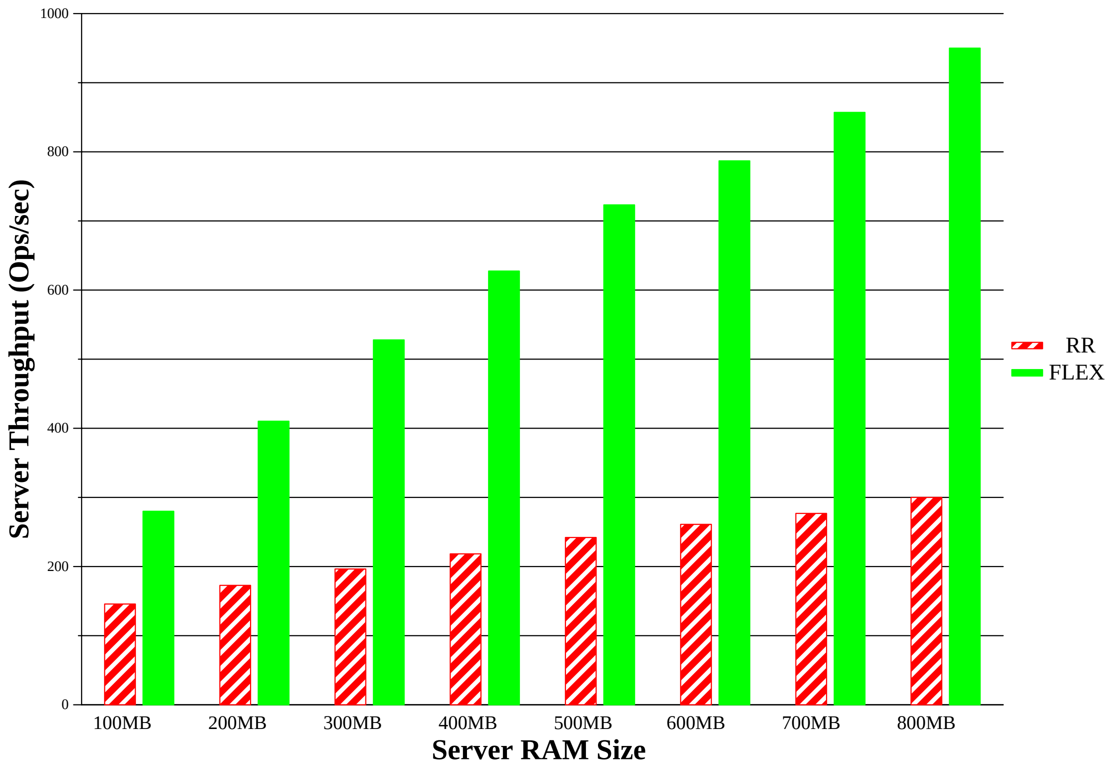

Figure 11: Server Throughput in the Cluster (Farm) of 8 Nodes under Round-Robin and FLEX Balancing Solutions.

Load balancing strategy FLEX outperforms the classic round-robin strategy by 2-3 times depending on a RAM size of the server.

The second trace was generated for 100 customers and 16 web servers, each with capacity of 1000 OPS, with *MaxCustSize = 30%*, and *RateBurstiness= 30*. The length of the trace was 40,000,000 requests.

The "customer profile" analysis and the results of FLEX balancing algorithm for this trace are shown in Appendix A.4.

The simulation results for this trace are shown in Figure 12.

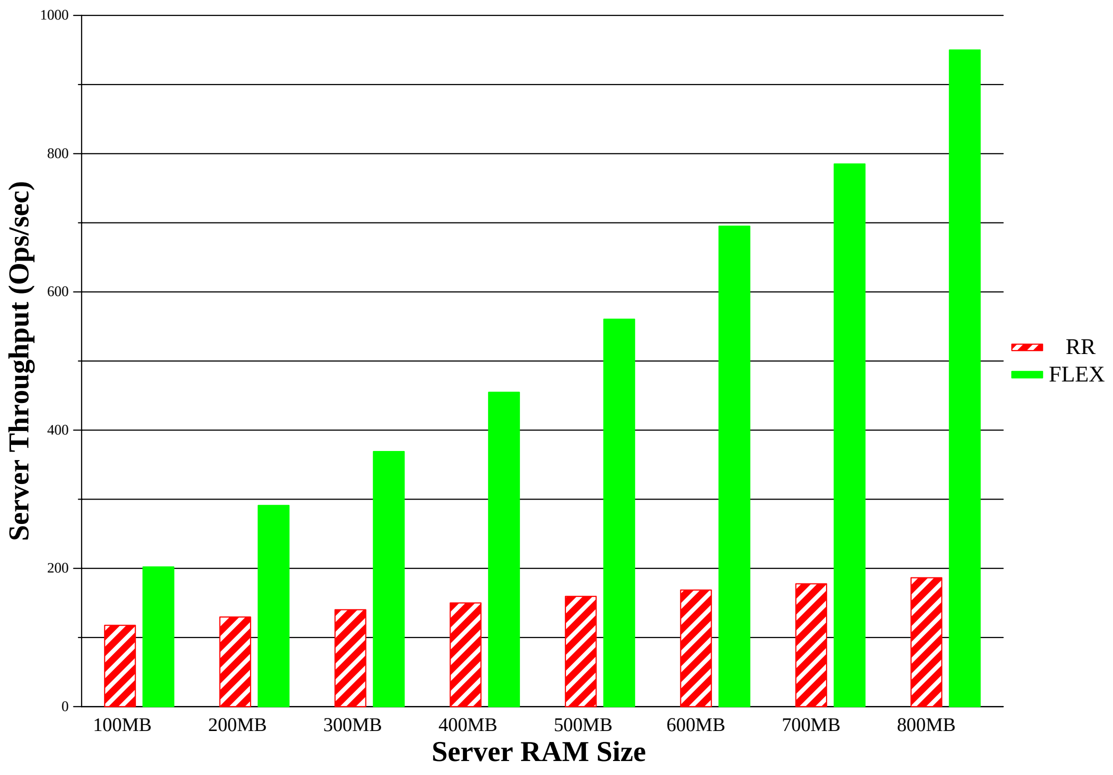

Figure 12: Server Throughput in the Cluster (Farm) of 16 Nodes under Round-Robin and FLEX Balancing Solutions.

Load balancing strategy FLEX outperforms the classic round-robin strategy by 2-5 times depending on a RAM size of the server.

10 Conclusion and Future Research

In this paper, we analyzed several hardware/software load-balancing solutions on the market which are used to distribute incoming stream of requests among a group of Web servers, and demonstrated their potential scalability problems. We outlined a new solution FLEX which provides a truly scalable performance.

A new solution FLEX (applied to a shared web hosting) acccomplishes a load-balancing task in a different way than the majority of trafitional solution. It can me considered as a planning and management tool for web hosting service.

It monitors and analyzes incoming requests over the period of time (analysis of web server logs can be one way to do it), and generates a favorable assignment of web sites to nodes in a cluster, and submits this information to a DNS server.

It allows to observe changing users' access behaviour, predict future scaling trends, and plan for it.

A key requirement in order to achieve scalability of the web server is that of balancing the load across the multiple front-end nodes and the back-end nodes. We are looking to extend the designed strategy to the management of the back-end nodes (or disks) as well.

The interesting future work will be to extend the solution and the algorithm to work with heterogenous nodes in a cluster, to take into account SLA (Service Level Agreement), and some additional QoS requirements.

We are planing to design an efficient hybrid router-DNS scheme which combines the logical router and web server functionality: the FLEX-DNS solution is used to get the coarse grained load balancing. Each web server node then acts as a router; if the load on the server is low, then the request is handled by that server, otherwise the router forwards the request to another node based on its load or global load balancing algorithm. This schema eliminates the forwarding by the router unless the load on the web server requires it.

11 References

[Alteon] URL: http://www.alteon.com/products/acelerate-data.html

[BrightTiger] URL: http://www.brighttiger.com

[Bruno97] L. Bruno: Balancing the Load On Web Servers. CMPnet, September 21, 1997. URL: http://www.data.com/roundups/balance.html

[Cisco] URL: http://www.cisco.com/warp/public/751/lodir/

[Ch99] L. Cherkasova: Scalable Web Hosting Service. HP Laboratories Report No. HPL-1999-52, April, 1999.

[F5Labs] URL: http://www.f5labs.com/

[FlyingFox] URL: http://www.flyingfox.com/

[HP-MCS] The HPL Multi-Computer Systems (MCS) Program. URL: http://www.hpl.hp.com/research/itc/csl/x86/

[HydraWEB] URL: http://www.hydraweb.com/z2_index.html

[IBM-SWND] http://www.software.ibm.com/network/dispatcher/

[IDC98] Web Hosting and E-Commerce Service Providers. Analysts: P. Burstyn, P. Johnson. International Data Corporation, Doc.No 16894, October 1998.

[LARD98] V.Pai, M.Aron, G.Banga, M.Svendsen, P.Drushel, W. Zwaenepoel, E.Nahum: Locality-Aware Request Distribution in Cluster-Based Network Servers. In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII), ACM SIGPLAN,1998, pp.205-216.

[MS97] S.Manley and M.Seltzer: Web Facts and Fantasy. Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997, pp.125-133.

[NSCA94] E. Katz, M. Butler, R.McGrath: A Scalable HTTP Server: The NSCA Prototype. Computer Networks and ISDN Systems, v.27, pp.155-163, 1994.

[NSCA95] T. Kwan, R. McGrath, D.Reed: NSCA's World Wide Web Server: Design and Performance. IEEE Computer, November 1995, pp.68-74.

[NSCA96] D. Dias, W. Kish, R. Mukherjee, R. Tewari: A Scalable and Highly Available Web Server. Proceedings of COMPCON'96, Santa Clara, 1996, pp.85-92.

[Resonate] URL: http://www.resonate.com/products/

[RND] URL: http://www.rndnetworks.com/

[Roberts98] E. Roberts: Load Balancing: On a Different Track. CMPnet, May 1998. URL: http://www.data.com/roundups/load.html

[RRDNS95] T. Brisco: DNS Support for Load Balancing.
RFC 1794, Rutgers University, April 1995.

[Schwetman95] Schwetman, H. Object-oriented simulation modeling with C++/CSIM. In Proceedings of 1995 Winter Simulation Conference, Washington, D.C., pp.529-533, 1995.

[SpecWeb96] The Workload for the SPECweb96 Benchmark.
URL http://www.specbench.org/osg/web96/workload.html

A APPENDIX

For readability and illustration reasons, in the following examples, we provide only the integer part of the numbers.

A.1 Example 1: 50 Customers Balanced Across 10 Servers

Example 1 is based on the following 50 customer's web sites and their traffic profiles:

Customer	WS	Rate(Normalized)
1	1	13
2	7	12
3	9	0
4	10	6
5	10	27
6	10	35
7	12	6
8	13	23
9	14	10
10	17	11
11	17	11
12	17	15
13	19	37
14	21	33
15	21	18
16	22	31
17	23	38
18	24	23
19	25	8
20	26	12
21	29	15
22	32	29
23	36	20
24	37	25
25	47	14
26	47	0
27	57	10
28	60	23
29	63	34
30	64	10
31	73	35
32	74	28
33	74	34
34	76	18
35	79	30
36	80	9
37	82	11
38	82	28
39	83	32
40	85	32
41	85	7
42	86	16
43	87	2
44	87	34
45	88	15
46	90	30
47	93	4
48	93	8
49	94	20
50	94	33

We performed one million iteration in the *flex-alpha* algorithm to partition the customers in 10 balanced groups. The best balanced partition was found on the iteration 236,698. The later iterations could not improve this result.

Accordingly to this partition (the Table serves as an input to construct the correspondent DNS Tables), the customers have to be assigned to the servers in the following way:

```
SERVER 0 CustNum  3 10 14 27 29 41
SERVER 1 CustNum  17 31 34 36
SERVER 2 CustNum  1 8 15 24 42 43
SERVER 3 CustNum  6 7 9 18 25 26 48
SERVER 4 CustNum  2 4 5 12 20 37 47
SERVER 5 CustNum  11 19 23 33 50
SERVER 6 CustNum  21 22 46 49
SERVER 7 CustNum  16 28 35 45
SERVER 8 CustNum  32 40 44
SERVER 9 CustNum  13 30 38 39
```

The partition, found by the *flex-alpha* algorithm, deviates from the “ideal” partition in the following way:

```
SERVER 0 TOTAL_RATE  98    TOTAL_WS 252  Even 247
SERVER 1 TOTAL_RATE  101   TOTAL_WS 252  Even 247
SERVER 2 TOTAL_RATE  101   TOTAL_WS 245  Even 247
SERVER 3 TOTAL_RATE  99    TOTAL_WS 247  Even 247
SERVER 4 TOTAL_RATE  90    TOTAL_WS 245  Even 247
SERVER 5 TOTAL_RATE  109   TOTAL_WS 246  Even 247
SERVER 6 TOTAL_RATE  95    TOTAL_WS 245  Even 247
SERVER 7 TOTAL_RATE  100   TOTAL_WS 249  Even 247
SERVER 8 TOTAL_RATE  95    TOTAL_WS 246  Even 247
SERVER 9 TOTAL_RATE  108   TOTAL_WS 248  Even 247
```

The combinations of working sets per server constituted by the working sets of assigned customers are the following:

```
SERVER 0  WS_Deviation  4   WS_Combination  9 17 21 57 63 85
SERVER 1  WS_Deviation  4   WS_Combination  23 73 76 80
SERVER 2  WS_Deviation  -2  WS_Combination  1 13 21 37 86 87
SERVER 3  WS_Deviation  0   WS_Combination  10 12 14 24 47 47 93
SERVER 4  WS_Deviation  -2  WS_Combination  7 10 10 17 26 82 93
SERVER 5  WS_Deviation  -1  WS_Combination  17 25 36 74 94
SERVER 6  WS_Deviation  -2  WS_Combination  29 32 90 94
SERVER 7  WS_Deviation  1   WS_Combination  22 60 79 88
SERVER 8  WS_Deviation  -1  WS_Combination  74 85 87
SERVER 9  WS_Deviation  0   WS_Combination  19 64 82 83
```

The combinations of the access rates per server constituted by the access rates of assigned customers are the following:

```
SERVER 0  RATE_Deviation  -1   RatesCombination  0 11 33 10 34 7
SERVER 1  RATE_Deviation  1    RatesCombination  38 35 18 9
SERVER 2  RATE_Deviation  1    RatesCombination  13 23 18 25 16 2
SERVER 3  RATE_Deviation  0    RatesCombination  35 6 10 23 14 0 8
SERVER 4  RATE_Deviation  -9   RatesCombination  12 6 27 15 12 11 4
SERVER 5  RATE_Deviation  9    RatesCombination  11 8 20 34 33
SERVER 6  RATE_Deviation  -4   RatesCombination  15 29 30 20
SERVER 7  RATE_Deviation  0    RatesCombination  31 23 30 15
SERVER 8  RATE_Deviation  -4   RatesCombination  28 32 34
SERVER 9  RATE_Deviation  8    RatesCombination  37 10 28 32
```

A.2 Example 2: 100 Customers Balanced Across 10 Servers

Example 2 is based on the following 100 customer's web sites and their traffic profiles:

Customer	WS	Rate(Normalized)
1	1	17
2	2	9
3	4	18
4	5	11
5	6	7
6	7	1
7	7	3
8	13	10
9	15	11
10	16	7
11	16	15
12	18	1
13	18	17
14	18	13
15	21	0
16	22	10
17	22	5
18	22	6
19	23	12
20	23	18
21	23	3
22	26	7
23	27	15
24	27	10
25	28	13
26	28	19
27	30	19
28	30	10
29	30	0
30	30	4
31	31	1
32	32	6
33	32	10
34	32	5
35	32	13
36	33	19
37	33	7
38	34	7
39	35	13
40	35	11
41	37	7
42	37	17
43	40	19
44	40	10
45	40	18
46	41	15
47	42	14
48	43	8
49	46	9
50	46	17
51	46	0
52	48	0
53	49	4
54	53	19
55	58	16
56	59	4

57	60	5
58	60	5
59	60	0
60	61	4
61	64	14
62	64	10
63	65	8
64	65	4
65	66	18
66	68	3
67	69	12
68	69	16
69	71	9
70	72	19
71	74	9
72	74	14
73	75	9
74	76	18
75	77	17
76	78	13
77	81	11
78	82	19
79	82	0
80	83	12
81	83	7
82	84	0
83	84	1
84	85	5
85	85	7
86	86	4
87	86	5
88	87	5
89	87	2
90	88	11
91	89	17
92	89	13
93	90	17
94	90	1
95	91	5
96	93	17
97	93	7
98	94	2
99	94	11
100	98	7

We performed one million iteration in the *flex-alpha* algorithm to partition the customers in 10 balanced groups. The best balanced partition was found on the iteration 148,067. The later iterations could not improve this result.

Accordingly to this partition (the Table serves as an input to construct the correspondent DNS Tables), the customers have to be assigned to the servers in the following way:

```
SERVER 0 CustNum  12 14 20 21 27 29 38 43 56 58 87 88
SERVER 1 CustNum  8 11 13 19 33 59 76 85 90 100
SERVER 2 CustNum  9 23 24 30 40 45 77 78 79 94
SERVER 3 CustNum  1 25 31 32 39 48 57 60 62 68 83
SERVER 4 CustNum  16 26 61 66 67 72 92 99
SERVER 5 CustNum  4 15 17 18 35 54 55 64 69 71 89
SERVER 6 CustNum  7 10 41 63 65 70 74 82 86
SERVER 7 CustNum  3 36 37 44 49 51 52 81 84 93
SERVER 8 CustNum  5 34 46 47 50 73 80 95 97
SERVER 9 CustNum  2 6 22 28 42 53 75 91 96 98
```

The partition, found by the *flex-alpha* algorithm, deviates from the "ideal" partition in the following way:

```
SERVER 0 TOTAL_RATE  102   TOTAL_WS 508  Even 508
SERVER 1 TOTAL_RATE  105   TOTAL_WS 511  Even 508
SERVER 2 TOTAL_RATE  105   TOTAL_WS 509  Even 508
SERVER 3 TOTAL_RATE  98    TOTAL_WS 508  Even 508
SERVER 4 TOTAL_RATE  100   TOTAL_WS 508  Even 508
SERVER 5 TOTAL_RATE  99    TOTAL_WS 510  Even 508
SERVER 6 TOTAL_RATE  88    TOTAL_WS 509  Even 508
SERVER 7 TOTAL_RATE  97    TOTAL_WS 508  Even 508
SERVER 8 TOTAL_RATE  95    TOTAL_WS 509  Even 508
SERVER 9 TOTAL_RATE  105   TOTAL_WS 504  Even 508
```

The combinations of the working sets per server constituted by the working sets of assigned customers are the following:

```
SERVER 0 WS_Deviation  0   WS_Combination  18 18 23 23 30 30 34 40 59 60 86 87
SERVER 1 WS_Deviation  2   WS_Combination  13 16 18 23 32 60 78 85 88 98
SERVER 2 WS_Deviation  0   WS_Combination  15 27 27 30 35 40 81 82 82 90
SERVER 3 WS_Deviation  0   WS_Combination  1 28 31 32 35 43 60 61 64 69 84
SERVER 4 WS_Deviation  0   WS_Combination  22 28 64 68 69 74 89 94
SERVER 5 WS_Deviation  1   WS_Combination  5 21 22 22 32 53 58 65 71 74 87
SERVER 6 WS_Deviation  0   WS_Combination  7 16 37 65 66 72 76 84 86
SERVER 7 WS_Deviation  0   WS_Combination  4 33 33 40 46 46 48 83 85 90
SERVER 8 WS_Deviation  0   WS_Combination  6 32 41 42 46 75 83 91 93
SERVER 9 WS_Deviation  -4  WS_Combination  2 7 26 30 37 49 77 89 93 94
```

The combinations of the access rates per server constituted by the access rates of assigned customers are the following:

```
SERVER 0  RATE_Deviation  2    RatesCombination  1 13 18 3 19 0 7 19 4 5 5 5
SERVER 1  RATE_Deviation  5    RatesCombination  10 15 17 12 10 0 13 7 11 7
SERVER 2  RATE_Deviation  5    RatesCombination  11 15 10 4 11 18 11 19 0 1
SERVER 3  RATE_Deviation  -1   RatesCombination  17 13 1 6 13 8 5 4 10 16 1
SERVER 4  RATE_Deviation  0    RatesCombination  10 19 14 3 12 14 13 11
SERVER 5  RATE_Deviation  0    RatesCombination  11 0 5 6 13 19 16 4 9 9 2
SERVER 6  RATE_Deviation  -11  RatesCombination  3 7 7 8 18 19 18 0 4
SERVER 7  RATE_Deviation  -2   RatesCombination  18 19 7 10 9 0 0 7 5 17
SERVER 8  RATE_Deviation  -4   RatesCombination  7 5 15 14 17 9 12 5 7
SERVER 9  RATE_Deviation  5    RatesCombination  9 1 7 10 17 4 17 17 17 2
```

A.3 Example 3: Synthetic Trace with 100 Customers Balanced Across 8 Servers

Example 3 is based on the synthetic trace. Here is their 100 customer's web sites and their traffic profiles constructed from the trace analysis:

	NORMALIZED	
Customer	WS	Rate
1	1	9
2	1	12
3	1	4
4	1	12
5	1	13
6	1	13

7	1	2
8	1	7
9	1	16
10	1	13
11	1	5
12	1	17
13	1	7
14	1	1
15	1	10
16	1	1
17	1	13
18	1	2
19	1	7
20	1	4
21	1	10
22	1	4
23	1	13
24	1	13
25	1	8
26	1	5
27	1	15
28	1	8
29	1	14
30	1	7
31	1	6
32	1	16
33	1	4
34	1	6
35	1	7
36	1	8
37	1	15
38	1	10
39	2	12
40	2	17
41	2	3
42	2	10
43	2	1
44	4	4
45	4	3
46	4	10
47	4	4
48	4	4
49	5	2
50	6	3
51	7	16
52	7	14
53	7	6
54	7	7
55	8	17
56	8	14
57	8	7
58	9	3
59	9	4
60	9	13
61	9	9
62	9	11
63	10	4
64	10	8
65	10	10
66	10	2
67	11	5
68	8	1

69	12	9
70	12	15
71	11	2
72	10	1
73	12	5
74	12	6
75	13	15
76	12	3
77	14	16
78	15	7
79	14	2
80	16	14
81	17	8
82	18	7
83	18	12
84	18	3
85	16	2
86	18	4
87	19	11
88	18	3
89	19	6
90	20	14
91	10	1
92	23	15
93	23	7
94	26	10
95	24	5
96	28	7
97	28	12
98	28	13
99	24	3
100	23	2

We performed one million iteration in the *flex-alpha* algorithm to partition the customers in 8 balanced groups. The best balanced partition was found on the iteration 926,932.

Accordingly to this partition (the Table serves as an input to construct the correspondent DNS Tables), the customers have to be assigned to the servers in the following way:

```
SERVER 0 CustNum  15 26 32 37 53 62 63 65 81 88 94
SERVER 1 CustNum  3 6 8 21 41 52 60 71 74 78 83 93
SERVER 2 CustNum  5 7 27 36 39 44 50 54 61 69 72 87 96
SERVER 3 CustNum  1 2 4 14 17 20 24 33 57 58 64 66 67 85 89 91
SERVER 4 CustNum  9 11 34 40 49 55 59 68 77 98 99
SERVER 5 CustNum  10 12 16 25 35 46 48 56 80 84 86 97
SERVER 6 CustNum  18 19 29 31 45 47 51 70 75 79 90 100
SERVER 7 CustNum  13 22 23 28 30 38 42 43 73 76 82 92 95
```

The partition, found by the *flex-alpha* algorithm, deviates from the "ideal" partition in the following way:

```
SERVER 0 TOTAL_RATE 97   TOTAL_WS  100  Even  100
SERVER 1 TOTAL_RATE 97   TOTAL_WS  100  Even  100
SERVER 2 TOTAL_RATE 100  TOTAL_WS  100  Even  100
SERVER 3 TOTAL_RATE 101  TOTAL_WS  100  Even  100
SERVER 4 TOTAL_RATE 99   TOTAL_WS  100  Even  100
SERVER 5 TOTAL_RATE 106  TOTAL_WS  100  Even  100
SERVER 6 TOTAL_RATE 99   TOTAL_WS  100  Even  100
SERVER 7 TOTAL_RATE 94   TOTAL_WS   99  Even  100
```

The combinations of the working sets per server constituted by the working sets of assigned customers are the following:

```
SERVER 0  WS_Deviation  0   WS_Combination  1 1 1 1 7 9 10 10 17 18 26
SERVER 1  WS_Deviation  0   WS_Combination  1 1 1 1 2 7 9 11 12 15 18 23
SERVER 2  WS_Deviation  0   WS_Combination  1 1 1 1 2 4 6 7 9 12 10 19 28
SERVER 3  WS_Deviation  0   WS_Combination  1 1 1 1 1 1 1 1 8 9 10 10 11 16 19 10
SERVER 4  WS_Deviation  0   WS_Combination  1 1 1 2 5 8 9 8 14 28 24
SERVER 5  WS_Deviation  0   WS_Combination  1 1 1 1 1 4 4 8 16 18 18 28
SERVER 6  WS_Deviation  0   WS_Combination  1 1 1 1 4 4 7 12 13 14 20 23
SERVER 7  WS_Deviation  -1  WS_Combination  1 1 1 1 1 1 2 2 12 12 18 23 24
```

The combinations of the access rates per server constituted by the access rates of assigned customers are the following:

```
SERVER 0  RATE_Deviation  -2   RatesCombination  9 4 15 14 5 10 3 9 7 2 9
SERVER 1  RATE_Deviation  -2   RatesCombination  3 12 6 9 2 13 12 1 5 6 11 6
SERVER 2  RATE_Deviation  0    RatesCombination  12 1 14 7 11 3 2 6 8 8 0 10 6
SERVER 3  RATE_Deviation  1    RatesCombination  8 11 11 0 12 3 12 3 6 2 7 1 4 1 5 0
SERVER 4  RATE_Deviation  0    RatesCombination  15 4 5 16 1 16 3 0 15 12 2
SERVER 5  RATE_Deviation  6    RatesCombination  12 16 0 7 6 9 3 13 13 2 3 11
SERVER 6  RATE_Deviation  0    RatesCombination  1 6 13 5 2 3 15 14 14 1 13 1
SERVER 7  RATE_Deviation  -5   RatesCombination  6 3 12 7 6 9 9 0 4 2 6 14 4
```

A.4 Example 4: Synthetic Trace with 100 Customers Balanced Across 16 Servers

Example 4 is based on the synthetic trace. Here is their 100 customer's web sites and their traffic profiles constructed from the trace analysis:

	NORMALIZED	
Customer	WS	Rate
1	1	23
2	1	4
3	1	7
4	1	30
5	1	24
6	1	26
7	1	26
8	1	8
9	1	8
10	1	11
11	1	12
12	1	16
13	2	8
14	2	14
15	2	28
16	2	2
17	2	9
18	3	12
19	4	18
20	4	11
21	5	26
22	5	15
23	5	3
24	6	8
25	6	29
26	6	31

27	7	10
28	7	29
29	7	8
30	7	12
31	7	24
32	7	15
33	7	29
34	8	13
35	8	18
36	8	31
37	9	11
38	9	13
39	9	6
40	11	7
41	12	12
42	12	14
43	11	2
44	12	22
45	12	2
46	14	18
47	14	10
48	15	9
49	15	17
50	14	2
51	17	30
52	17	22
53	17	26
54	19	5
55	20	23
56	20	23
57	20	27
58	21	20
59	21	30
60	21	15
61	21	16
62	14	1
63	23	20
64	18	2
65	24	19
66	26	14
67	26	29
68	27	10
69	27	10
70	28	27
71	28	24
72	24	3
73	28	11
74	28	14
75	29	31
76	29	20
77	31	26
78	31	15
79	31	25
80	22	2
81	28	5
82	31	18
83	31	29
84	31	28
85	31	12
86	30	14
87	31	27
88	30	9

89	26	4
90	31	19
91	31	25
92	28	5
93	31	26
94	31	24
95	30	13
96	31	16
97	30	12
98	31	14
99	16	1
100	31	17

We performed one million iteration in the *flex-alpha* algorithm to partition the customers in 16 balanced groups. The partition found on the iteration 402,080 was the best. The later iterations could not improve this result.

Here is the partition (i.e. how the customers have to be assigned to the servers). This Table serves as input to construct the correspondent DNS Tables.

```
SERVER 0 CustNum  44 70 75 77
SERVER 1 CustNum  14 27 32 34 47 79 82
SERVER 2 CustNum  1 3 16 18 20 29 60 85 97
SERVER 3 CustNum  9 46 51 54 59 74
SERVER 4 CustNum  38 76 83 96
SERVER 5 CustNum  4 15 43 50 58 68 72
SERVER 6 CustNum  36 84 88 90
SERVER 7 CustNum  17 25 86 93 94
SERVER 8 CustNum  2 35 41 55 71 100
SERVER 9 CustNum  5 6 13 40 66 73 87
SERVER 10 CustNum  30 31 37 56 61 62 63
SERVER 11 CustNum  7 8 12 42 81 89 91
SERVER 12 CustNum  19 22 23 45 48 52 57 80
SERVER 13 CustNum  21 24 28 33 64 69 98
SERVER 14 CustNum  10 26 39 67 92 95
SERVER 15 CustNum  11 49 53 65 78 99
```

The partition, found by the *flex-alpha* algorithm, deviates from the "ideal" partition in the following way:

```
SERVER 0  TOTAL_RATE 105   TOTAL_WS  100  Even  100
SERVER 1  TOTAL_RATE 104   TOTAL_WS  100  Even  100
SERVER 2  TOTAL_RATE 101   TOTAL_WS  100  Even  100
SERVER 3  TOTAL_RATE 104   TOTAL_WS  100  Even  100
SERVER 4  TOTAL_RATE 77    TOTAL_WS  100  Even  100
SERVER 5  TOTAL_RATE 94    TOTAL_WS  100  Even  100
SERVER 6  TOTAL_RATE 86    TOTAL_WS  100  Even  100
SERVER 7  TOTAL_RATE 101   TOTAL_WS  100  Even  100
SERVER 8  TOTAL_RATE 97    TOTAL_WS  100  Even  100
SERVER 9  TOTAL_RATE 116   TOTAL_WS  100  Even  100
SERVER 10 TOTAL_RATE 106   TOTAL_WS  101  Even  100
SERVER 11 TOTAL_RATE 97    TOTAL_WS  100  Even  100
SERVER 12 TOTAL_RATE 97    TOTAL_WS  100  Even  100
SERVER 13 TOTAL_RATE 117   TOTAL_WS  101  Even  100
SERVER 14 TOTAL_RATE 94    TOTAL_WS  100  Even  100
SERVER 15 TOTAL_RATE 89    TOTAL_WS  104  Even  100
```

The combinations of the access rates per server constituted by the access rates of assigned customers are the following:

```
SERVER 0   WS_Deviation  0   WS_Combination   12 28 29 31
SERVER 1   WS_Deviation  0   WS_Combination   2 7 7 8 14 31 31
SERVER 2   WS_Deviation  0   WS_Combination   1 1 2 3 4 7 21 31 30
SERVER 3   WS_Deviation  0   WS_Combination   1 14 17 19 21 28
SERVER 4   WS_Deviation  0   WS_Combination   9 29 31 31
SERVER 5   WS_Deviation  0   WS_Combination   1 2 11 14 21 27 24
SERVER 6   WS_Deviation  0   WS_Combination   8 31 30 31
SERVER 7   WS_Deviation  0   WS_Combination   2 6 30 31 31
SERVER 8   WS_Deviation  0   WS_Combination   1 8 12 20 28 31
SERVER 9   WS_Deviation  0   WS_Combination   1 1 2 11 26 28 31
SERVER 10  WS_Deviation  0   WS_Combination   7 7 9 20 21 14 23
SERVER 11  WS_Deviation  0   WS_Combination   1 1 1 12 28 26 31
SERVER 12  WS_Deviation  0   WS_Combination   4 5 5 12 15 17 20 22
SERVER 13  WS_Deviation  0   WS_Combination   5 6 7 7 18 27 31
SERVER 14  WS_Deviation  0   WS_Combination   1 6 9 26 28 30
SERVER 15  WS_Deviation  3   WS_Combination   1 15 17 24 31 16
```

The combinations of the access rates per server constituted by the access rates of assigned customers are the following:

```
SERVER 0   RATE_Deviation  5     RatesCombination  21 26 30 25
SERVER 1   RATE_Deviation  4     RatesCombination  13 9 14 12 9 24 17
SERVER 2   RATE_Deviation  1     RatesCombination  22 6 1 11 10 7 14 11 11
SERVER 3   RATE_Deviation  4     RatesCombination  7 17 29 4 29 13
SERVER 4   RATE_Deviation  -22   RatesCombination  12 19 28 15
SERVER 5   RATE_Deviation  -5    RatesCombination  29 27 1 1 19 9 2
SERVER 6   RATE_Deviation  -13   RatesCombination  30 27 8 18
SERVER 7   RATE_Deviation  1     RatesCombination  8 28 13 25 23
SERVER 8   RATE_Deviation  -2    RatesCombination  3 17 11 22 23 16
SERVER 9   RATE_Deviation  16    RatesCombination  23 25 7 6 13 10 26
SERVER 10  RATE_Deviation  6     RatesCombination  11 23 10 22 15 0 19
SERVER 11  RATE_Deviation  -2    RatesCombination  25 7 15 13 4 3 24
SERVER 12  RATE_Deviation  -2    RatesCombination  17 14 2 1 8 21 26 1
SERVER 13  RATE_Deviation  17    RatesCombination  25 7 28 28 1 9 13
SERVER 14  RATE_Deviation  -5    RatesCombination  10 30 5 28 4 12
SERVER 15  RATE_Deviation  -10   RatesCombination  11 16 25 18 14 0
```