FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service

Ludmila Cherkasova
Hewlett-Packard Labs
1501 Page Mill Road,Palo Alto, CA 94303, USA
e-mail: {cherkasova}@hpl.hp.com

Abstract

FLEX is a new scalable "locality aware" solution for achieving both load balancing and efficient memory usage on a cluster of machines hosting several web sites. FLEX allocates the sites to different machines in the cluster based on their traffic characteristics. This aims to avoid the unnecessary document replication to improve the overall performance of the system. The desirable routing can be done by submitting the corresponding 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 Round-Robin based solutions and FLEX over the range of different workloads. For generated traces, FLEX outperforms Round-Robin based solutions 2-5 times.

1 Introduction

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 are looking for a cost efficient, "no hassle" solution. A shared Web hosting service creates a set of virtual servers on the same server. This supports the illusion that each host has its own web server, when in reality, multiple "logical hosts" share one physical host.

Traditional load balancing 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 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 might effectively have almost the same RAM as one node, because of the replicated popular content.

These observations 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. For each web site hosted on a cluster, FLEX evaluates (using web server access logs) the system resource requirements in terms of the memory (site's working set) and the load (site's access rate). The sites are then partitioned into N balanced groups based on their memory and load requirements and assigned to the N nodes of the cluster respectively. Since each hosted web site has a unique domain name, the desired routing of requests is achieved by submitting appropriate configuration files to the DNS server.

One of the main attractions of this approach is its ease of deployment. This solution requires no special hardware support or protocol changes. There is no single front end routing component. Such a component can easily become a bottleneck, especially if content based routing requires it to do such things as tcp connection hand-offs etc. FLEX can be easily implemented on top of the current infrastructure used by Web hosting service providers.

2 Shared Web Hosting: Typical Solutions

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

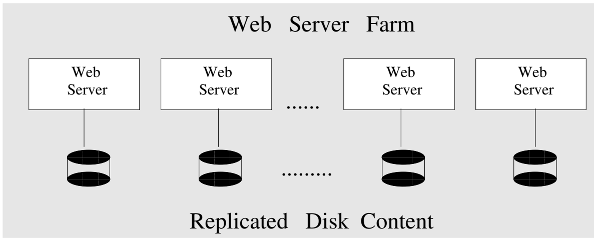

Figure 1: Web Server Farm with Replicated Disk Content.

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. 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 and back-end nodes are mapped to different physical

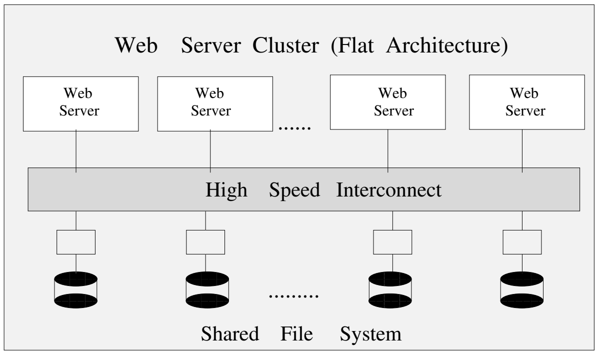

Figure 2: Web Server Cluster (Flat Architecture).

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 nodes. The NSCA prototype of the scalable HTTP server based on two-tier architecture is described and studied in [NSCA96].

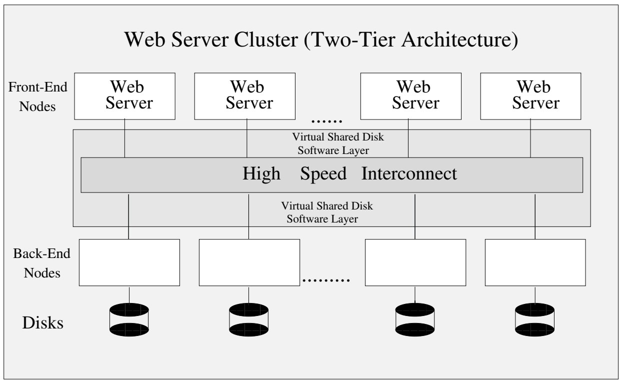

Figure 3: Web Server Cluster (Two Tier Server Architecture).

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

3 Load Balancing 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.

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. R-R 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. In most of

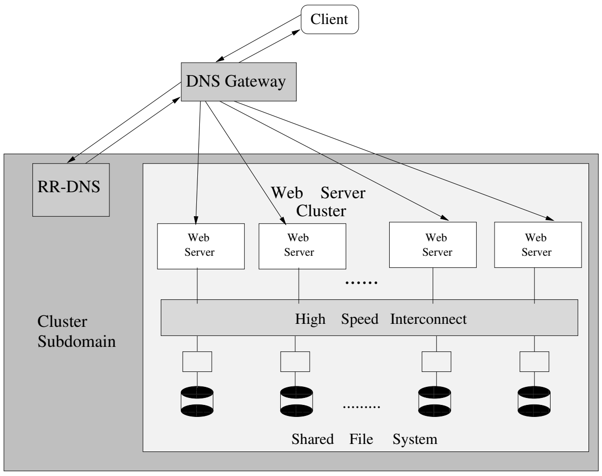

Figure 4: Web Server Cluster Balanced with Round-Robin DNS.

the cases, R-R 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.

3.2 IP/TCP/HTTP Redirection Based Approaches

The market now offers several hardware/software load-balancer solutions.

Hardware 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

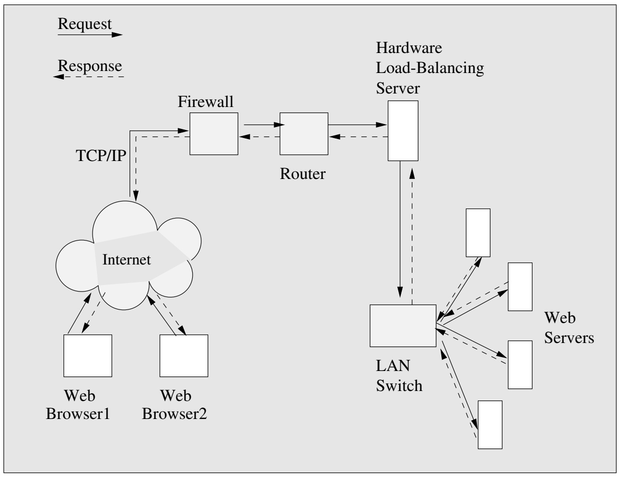

Figure 5: Web Server Farm with Hardware Load-Balancing Server.

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 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. 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. Both inbound requests and outbound responses must pass through the balancing server, causing a load-balancer to become a potential bottleneck.

Four of the six hardware load balancers on the market are built around Intel pentium processors: LocalDirector from Cisco Systems, Fox Box from Flying Fox, BigIP from F5 Labs, and Load Manager 1000 from Hydraweb Technologies Inc. Another two load balancers employ a RISC chip: Web Server Director from RND Networks Inc. and ACEdirector from 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.

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. Actually, that

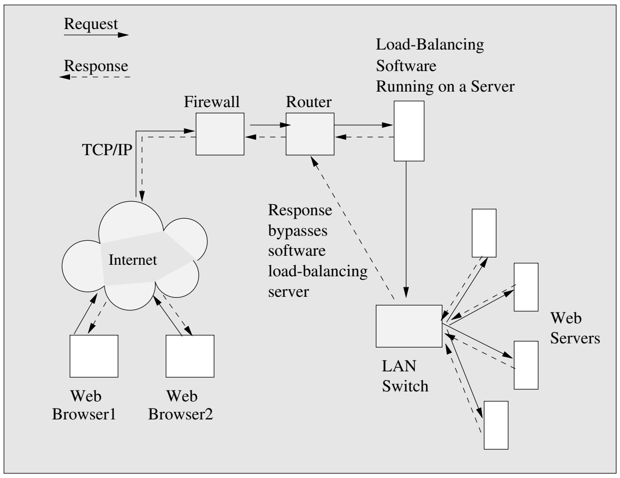

Figure 6: Web Server Farm with Load-Balancing Software Running on a Server.

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.

Three software load-balancing servers are available: ClusterCATS from Bright Tiger Technologies, SecureWay Network Dispatcher from IBM, and Central Dispatch from Resonate Inc. These products are loaded onto Unix or Windows NT servers.

3.3 "Locality-Aware" Balancing Strategies

Traditional load balancing solutions (both hardware and software) try to distribute the requests uniformly on all the machines in a cluster. However, this adversely affects efficient memory usage because content is replicated across the caches of all the machines.[1] This may significantly decrease overall system performance.

This observation have led to a design of the new "locality-aware" request distribution strategy (LARD) which was 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 throughput by a factor of two to four for 16 nodes cluster.

4 New Scalable Web Hosting Solution: FLEX

FLEX motivation is similar to the "locality-aware" balancing strategy discussed above: to avoid the unnecessary document replication to improve the overall system performance. 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 web sites as a starting point. Each of these sites has different traffic patterns in terms of the accessed files (memory requirements) and the access rates (load requirements).

Let S be a number of sites hosted on a cluster of N web servers. For each web site s, we build the initial "site profile" SP_s by evaluating the following characteristics:

- $AR(s)$ - the access rate to the content of a site s (in *bytes* transferred during the observed period P);
- $WS(s)$ - the combined size of all the accessed files of site s (in *bytes* during the observed period P, so-called "working set");

This site profile is entirely based on information which can be extracted from the web server access logs of the sites.

The next step is to partition all the sites in N "equally balanced" groups: $S_1, ..., S_N$ in such a way, that combined access rates and combined "working sets" of sites in each of those S_i are approximately the same. We designed a special algorithm *flex-alpha* which does it (see Section 5). The final step is to assign a server N_i from a cluster to each group S_i.

The FLEX solution is deployed by providing the corresponding information to a DNS server via configuration files. In such a way, the site domain name is resolved to a corresponding IP address of the assigned node (nodes) in the cluster. This solution is flexible and easy to manage. Tuning can be done on a daily or weekly basis. If server logs analysis shows enough changes, and the algorithm finds a better partitioning of the sites to the nodes in the cluster, then new DNS configuration files are generated. Once DNS server has updated its configuration tables,[1] new requests are routed accordingly to a new configuration files, and this leads to more efficient

[1] We are interested in the case when the overall file set is greater than the RAM of one node. If the entire file set completely fits to the RAM of a single machine, any of existing load balancing strategies provides a good solution.

[1] The entries from the old configuration tables can be cached by some servers and used for request routing without going to primary DNS server. However, the cached entries are valid for a limited time only dictated by TTL (*time to live).* Once TTL is expired, the primary 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 an access to the whole content and can satisfy any request.

traffic balancing on a cluster. The logic of the FLEX strategy is shown in Figure 7. Such a self-monitoring solution helps

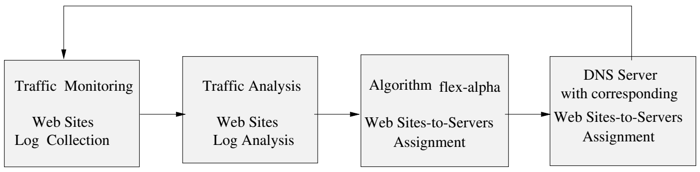

Figure 7: FLEX Strategy: Logic Outline.

to observe changing users' access behaviour and to predict future scaling trends.

5 Load Balancing Algorithm *flex-alpha*

We designed a special algorithm, called *flex-alpha* to partition all the sites in the "equally balanced" groups by the number of the nodes in a cluster. Each group of sites is served by the assigned server in a cluster. We will call such an assignment as *partition.* We use the following notations:

- *NumSites* – a number of sites hosted on a web cluster.
- *NumServers* – a number of servers in a web cluster.
- *SiteWS[i]* – an array which provides the combined size of the requested files of the i-th site, so-called "a working set" for the i-th site. We assume that the sites are ordered by the working set, i.e. the array *SiteWS[i]* is ordered.
- *SiteAR[i]* – an array which provides the access rates to the i-th site, i.e. *all the bytes* requested of the i-th site.

At first, we are going to normalize the working sets and the access rates of the sites.

$$WorkingSetTotal = \sum_{i=1}^{NumSites} SiteWS[i]$$

$$RatesTotal = \sum_{i=1}^{NumSites} SiteAR[i]$$

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

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

Now, the overall goal can be rephrased in the following way: we aim to partition all the sites in *NumServers* "equally balanced" groups: $S_1, ..., S_N$ in such a way, that

- cumulative "working sets" in each of those S_i groups are close to 100%, and
- cumulative "access rates" in each of those S_i groups are around 100%.

The pseudo-code[2] of the algorithm *flex-alpha* is shown below in Figure 8. We use the following notations:

- *SitesLeftList* – the ordered list of sites which are not yet assigned to the servers. In the beginning, the *SitesLeftList* is the same as the original ordered list of sites *SitesList*;
- *ServerAssignedSites[i]* – the list of sites which are assigned to the i-th server;
- *ServerWS[i]* – the cumulative "working set" of the sites currently assigned to the i-th server;
- *ServerAR[i]* – the cumulative "access rate" of the sites currently assigned to the i-th server.
- $dif(x, y)$ – the absolute difference between x and y, i.e. $(x - y)$ or $(y - x)$, whatever is positive.

Assignment of the sites to the servers (except the last one) is done accordingly to the pseudo-code in Figure 8. Fragment of the algorithm shown in Figure 8 is applied in a cycle to the the first *NumServers*-1 servers.

```
/* we assign sites to the i-th server from the
 * SitesLeftList using random function until the
 * addition of the chosen site content  does not
 * exceed the ideal content limit per server 100%.
 */
site = random(SitesLeftList);
if (ServerWS[i] + SiteWS[site]) <= 100){
   append(ServerAssignedSites[i], site);
   remove(SitesLeftList, site);
   ServerWS[i] = ServerWS[i] + SiteWS[site];
   ServerAR[i] = ServerAR[i] + SiteAR[site];
}
else {
/* if the addition of the chosen site content
 * exceeds the ideal content limit per server
 * 100%  we try to find such a Site from the
 * SitesLeftList which results in a minimum
 * deviation from the SpaceLeft on this server.
 */
  SpaceLeft = 100 - ServerWS[i];
  find Site with min(dif(SpaceLeft - SiteWS[Site]));
  append(ServerAssignedSites[i], Site);
  remove(SitesLeftList, Site);
  ServerWS[i] = ServerWS[i] + SiteWS[Site];
  ServerAR[i] = ServerAR[i] + SiteAR[Site];
}
if (ServerWS[i]) > 100) {
/* small optimization at the end: returning the
 * sites with smallest working sets (extra_site)
 * back to the SitesLeftList until the deviation
 * between the server working set ServerWS[i] and
 * the ideal content per server 100% is minimal.
 */
  if (dif(100 - (ServerWS[i] - SiteWS[extra_site]))<
     dif(100 - (ServerWS[i])) {
     append(SitesLeftList, extra_site);
     remove(ServerAssignedSites[i], extra_site);
     ServerWS[i] = ServerWS[i] + SiteWS[extra_site];
     ServerAR[i] = ServerAR[i] + SiteAR[extra_site];
  }
}
```

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

All the sites which are left in *SitesLeftList* are assigned to the last server. This completes one iteration of the algorithm, resulting in the assignment of all the sites to the servers in balanced groups. Typically, this algorithm generates a very

[2] We describe the basic case only. For exceptional situation, when some sites have their working sets larger than 100%, the advanced algorithm to address this situation is designed in [CP00].

good balanced partition with respect to the 'working sets" of the sites assigned to the servers.

The second goal is to balance the cumulative access rates per server. For this purpose, for each partition P generated by the algorithm, the rate deviation of P is computed:

$$RateDev(P) = \sum_{i=1}^{NumServers} dif(100, ServerRate[i])$$

We define partition P_1 is *better rate-balanced* than partition P_2 iff $RateDev(P_1) < RateDev(P_2)$.

The algorithm *flex-alpha* 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 in 10,000 - 100,000 iterations.

6 Synthetic Trace Generator

We developed a synthetic trace generator to evaluate the performance of FLEX. There is a set of basic parameters which defines the traffic pattern, file distribution, and "web site" profiles in generated synthetic trace:

1. *NumSites* - number of web sites sharing the cluster;

2. *NumWebServers* - number of web servers in the cluster. This parameter is used to define a number of files directories in the content. We use a simple scaling rule: trace targeted to run on N-nodes cluster has N times greater number of directories than a 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 1Kb-9Kb (with access rate of 50%), *2 class* files are 10Kb-90Kb (with access rate of 14%), *3 class* files are 100Kb-900Kb (with access rate of 1%).

4. *MaxSiteSize* - a desirable maximum size of normalized working set per web site. The *MaxSiteSize* is used as an additional constraint: $SiteWS[i] \leq MaxSiteSize$.

5. *RateBurstiness* - a range for a number of consequent requests to the same web site.

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

Synthetic traces allow to create different traffic patterns, different files distributions, and different "web site" profiles. This variety is useful to evaluate the FLEX strategy over wide variety of possible workloads. Evaluation of the strategy for the real web hosting service is a next step in our research.

7 Simulation Results with Synthetic Traces

We built high level simulation model of web cluster (farm) using C++Sim [Schwetman95]. The model makes the following assumptions about the capacity of each web server in the cluster:

- Web server throughput is 1000 Ops/sec when retrieving files of size 14.5K from the RAM (14.5k is the average file size for the SpecWeb96 benchmark).
- Web server throughput is 10 times lower when it retrieves the files from the disk, [1](i.e., 100 Ops/sec)
- The service time for a file is proportional to the file size.
- The cache replacement policy is LRU.

The first trace was generated for 100 sites and 8 web servers, with *MaxSiteSize = 30%*, and *RateBurstiness = 30*. The length of the trace was 20 million requests.

The second trace was generated for 100 sites and 16 web servers, with *MaxSiteSize = 30%*, and *RateBurstiness = 30*. The length of the trace was 40 million requests.

Each trace was analyzed, and for each web site s, the corresponding "site profile" SP_s was built. After that, using *flex-alpha* algorithm, a partition was generated for each trace and its web sites. The requests from the first (second) original trace were split into eight (sixteen) sub-traces based on the strategy. The eight (sixteen) sub-traces were then "fed" to the respective servers. Each server picks up the next request from its sub-trace as soon as it is finished with the previous request. We measured two metrics: *server throughput* (averaged across all the servers), and the *miss ratio.*

The simulation results for the first trace (throughput and miss ratio) are shown in Figure 8, 9.

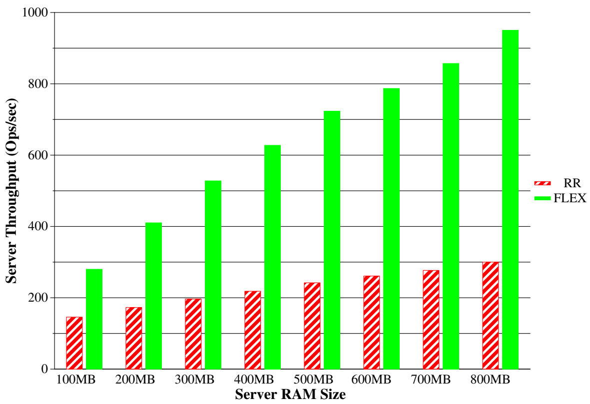

Figure 8: Server Throughput in the Cluster of 8 Nodes.

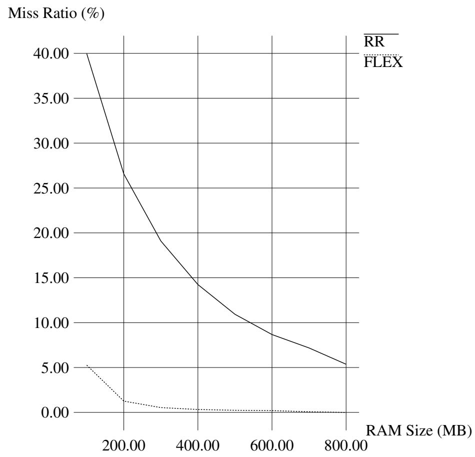

Figure 9: Average Miss Ratio in the Cluster of 8 Nodes.

[1] We measured web server throughput (on HP 9000/899 running HP-UX 11.00) when it supplied files from the RAM (i.e., the files were already downloaded from disk and resided in the File Buffer Cache), and compared it against the web server throughput when it supplied files from the disk. Difference in throughput was a factor of 10. For machines with different configurations, this factor can be different).

Server throughput is improved 2-3 times with FLEX load balancing strategy against the classic round-robin strategy. Miss ratio improvement is even higher: 5-8 times. These results deserve some explanation. Accordingly to our "partitioning" and the SpecWeb'96 requirements: the total working set of the sites assigned to one web server is 750MB. If a web server has a RAM of 750MB or larger then all the files are eventually brought to a RAM, and all the consequent requests are satisfied from RAM, resulting in the best possible server throughput. By "partitioning" the sites across the cluster, FLEX is able to achieve the best performance for RAM=800MB with nearly zero miss ratio, because all the files for all the sites reside in a RAM of the assigned servers. Round Robin strategy, however, is dealing with total working set of 750MB x 8 (8 is a number of servers in this simulation), and has a miss ratio of 5.4%. As a corollary, a server throughput is 3 times worse.

The simulation results for the second trace (throughput and miss ratio) are shown in Figure 10, 11.

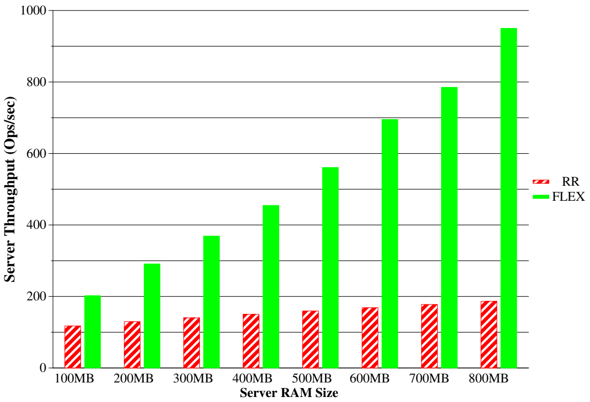

Figure 10: Server Throughput in the Cluster of 16 Nodes.

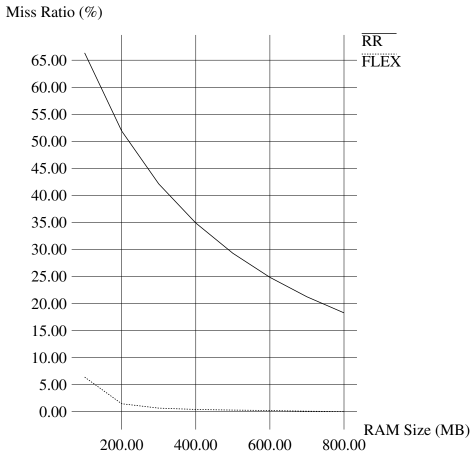

Figure 11: Average Miss Ratio in the Cluster of 16 Nodes.

Server throughput is improved 2-5 times with FLEX load balancing strategy against the classic round-robin strategy. Miss ratio improvement is even higher than in previous case. Explanations are similar to the case with 8 servers. By "partitioning" the sites across the cluster, FLEX is able to achieve the best possible server performance for the RAM=800MB, because all the files for all the sites reside in a RAM of the assigned servers. Round Robin strategy, however, is dealing with total working set of 750MB x 16 (16 is a number of servers in this simulation), and has a miss ratio of 18%. As a corollary, a server throughput is 5 times worse.

Note, that FLEX strategy shows a scalable performance: the results for 16 servers are only slightly worse compared against FLEX results for 8 servers. Round robin strategy performance is clearly worse for 16 servers against 8 servers case: server throughput is 19-33% worse, and miss ratio increases 0.6-3 times.

8 Conclusion and Future Research

In this paper, we analyzed several load-balancing solutions on the market, and demonstrated their potential scalability problem. We introduced a new "locality-aware" balancing solution FLEX, and analyzed its performance. The benefits of the FLEX can be summarized as follows:

- FLEX is a cost-efficient balancing solution. It does not require installation of any additional software. From analysis of the server logs, FLEX generates a favorable sites assignement to the servers, and forms configuration information for a DNS server.
- FLEX is a self-monitoring solution. It allows to observe changing users' access behaviour and to predict future scaling trends, and plan for it.
- FLEX is truly scalable solution.
- FLEX allows to save an additional hardware by more efficient usage of available resources. It could outperform current market solutions up to 2-5 times.

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.

References

[C99] L. Cherkasova: FLEX: Design and Management Strategy for Scalable Web Hosting Service. HP Labs Report, No. HPL-1999-64R1,1999.

[CP00] L. Cherkasova, S. Ponnekanti: Achieving Load Balancing and Efficient Memory Usage in A Web Hosting Service Cluster. HP Labs Report No. HPL-2000-27, 2000.

[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 ASPLOS-VIII, ACM SIGPLAN,1998, pp.205-216.

[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.

[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, pp.529-533, 1995.

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