Kernel Implementations of Locality-Aware Dispatching Techniques for Web Server Clusters

Michele Di Santo, Nadia Ranaldo, Eugenio Zimeo
RCOST- University of Sannio, 82100 Benevento, Italy
zimeo@unisannio.it

IEEE Cluster Computing 03
Hong Kong
2 December 2003
Introduction

- The growth of the Web has driven a high demand for powerful web servers
  - End-users should perceive that they are using very responsive services
  - Unfortunately, this goal cannot be easily achieved when a large amount of requests are directed to few popular Web providers
- Solutions that can interest providers should be based on low-cost equipment
  - No changes to the network architecture of the Internet and to Web applications
Many proposed solutions aim at ensuring scalability and availability by using clusters of computers.

- These clusters are viewed as unique virtual Web servers at the client-side.

**Scalability** is achieved by distributing the load among the computers of the clusters.

However, selecting a server in a cluster only on the basis of its load condition can be unsatisfactory.

- Modern c/s applications often require secure and stateful transactions.
- These transactions cannot be guaranteed if a content-blind dispatching scheme is adopted.

An increasingly popular technique for selecting a server of a cluster is based on the request content.
Content-aware dispatching: benefits

- This technique allows each HTTP request to be dispatched toward the “right” real server

- The meaning of the term “right” depends on the objectives that are to be achieved:
  - to support data integrity when SSL is used;
  - to guarantee stateful transactions, as the ones carried out in e-commerce environments;
  - to increase performance, by improving hit/miss ratios in the disk cache of real servers;
  - to achieve a high scalability of secondary storage by partitioning Web documents among different real servers of the cluster;
  - to use heterogeneous and specialized hardware for handling different HTTP requests, such as CGI execution, image retrieval, and so on.
Content at a glance

- We are principally interested in obtaining a small response time by increasing the hit/miss ratio in the disk cache of real servers
  - the other objectives will be pursued in the future work
- Content table:
  - An overview of forwarding mechanisms
  - Description of TCP modifications to support content-aware scheduling at kernel level
  - Some implementation issues of a content-aware scheduling algorithm atop of the modified TCP
  - Performance evaluations
  - A hybrid locality-aware algorithm based on cache prediction
  - Conclusions
Content-aware dispatching: solutions

- Content-aware scheduling algorithms need new techniques to enable the dispatching of HTTP requests.
- A simple approach consists of using a HTTP server as a dispatcher of requests:
  - all the requests are analyzed and managed in the user space;
  - a request message must cross all the OS layers to reach the user space, and then go down again toward the network.
- A more efficient solution requires that scheduling is implemented at the kernel level of the web switch OS:
  - dispatching of TCP segments is easy for content-blind scheduling algorithms;
  - content-aware dispatching is made difficult by:
    - the connection oriented nature of TCP;
    - the “three-way handshake”
Dispatching at kernel level

- With the content-aware dispatching, the real server can be selected only after the three-way handshake has been completed
  - A client sends the first part of a HTTP request only after having received the ACK for its SYN segment
- To solve the problem, two solutions can be adopted:
  - avoiding the three-way-handshake;
  - delaying scheduling in order to wait for the arrival of the TCP segment containing the HTTP request
- The former approach could be implemented by using T/TCP
  - it needs changes, both to clients and servers, that have a strong impact on the World-Wide Web infrastructure
- The latter approach requires changes to the OS of the web switch in order to delay the forwarding of packets toward the servers until a HTTP request arrives
Delaying the forwarding of packets

- Two are the most used solutions:
  - TCP-splicing
  - TCP-handoff

- **TCP-splicing**: two connections are established
  - One between the client and the web switch and another between the web switch and a real server
  - The connection can be spliced by using the NAT

- **TCP-handoff**: one connection is established
  - The web switch establishes a connection with the client and then transfers its TCP state to the selected real server in the cluster
TCP-handoff: our solution

- The key idea consists in forcing the selected real server to adopt the Initial Sequence Number (ISN) chosen by the web switch during the three-way-handshake with the client.
- This way, the real server can directly reply TCP segments to the client:
  - Reply segments have to not cross the dispatcher.
  - Sequence Numbers (SNs) are not to be rewritten.
- Segment routing is based on the direct forwarding of packets from the web switch to real servers:
  - MAC addresses re-writing or encapsulation.
- However, this approach requires the OS kernel of computers hosting real servers to be modified:
  - To make our approach portable, we have introduced a new option at TCP level, called fast connection option.
TCP-handoff: our solution

SYN (x)
SYN (y)
ACK (x+1)
ACK (y+1)
HTTP Request (x+1)

SYN (HTTP Request, x)
ACK
ACK
FIN
ACK
ACK
FIN
ACK

LISTEN
SYN_RCVD
ESTABLISHED
CLOSE_WAIT
LISTEN

Timeout

instead of

HTTP Request
ACK
Our implementation is based on the Linux Virtual Server (LVS)

LVS uses the IP masquerade mechanism in the kernel 2.2.x and the netfilter framework in the kernel 2.4.x

Currently we have modified the IP masquerade mechanism
  - A TCP hand-off implementation based on netfilter for the kernel 2.4.x is under development

LVS provides some forwarding techniques
  - NAT, tunneling and direct routing

... and supports several content-blind scheduling algorithms for each service
  - Round Robin (RR), Least Connections (LC), ...

The Linux kernel was modified in order to implement our TCP hand-off variant

The set of LVS scheduling algorithms was extended by implementing a name-based algorithm
Implementation of a scheduling algorithm

• The mechanism proposed in the previous section has been tested by implementing the *Highest Random Weight* scheduling algorithm.

• The algorithm tries to achieve two main objectives:
  - minimizing the scheduling overhead
  - minimizing the latency perceived by users

• HRW is based on a hashing technique that can select a real server by using a hash value associated to the URL \((n)\) contained in a HTTP request:

\[
F(n) = \text{\textit{Si : } } W(n, Si) \geq W(n, Sj) \quad i, j=1,2,..., m \quad \text{and } i \neq j
\]

• Due to the hash-based mapping, HRW can always associate the same real server to a \(n\):
  - **Increases the hit/miss ratio** in the real server disk cache

• The HRW mapping is not static; it is able to select a different real server if the one previously bound to a request has crashed.
A reference measurement of latency was obtained by using a direct connection between the client and the server.

Other measurements of latency have been obtained by using the web switch both for content-aware and content-blind routing.

- Content-blind scheduling delay is 38 us
- Content-aware scheduling delay is 87 us
- The routing overhead for each acknowledgement crossing the web switch is 17 us
Performance evaluation: best case

- Three identical requests are directed to the distributed Web server
  - HRW - all the requests are dispatched to the same real server, which can consequently load the requested object from the disk cache (1 miss and 2 hits)
  - RR and LC - the requests are dispatched to three different real servers, which are consequently forced to load objects from the disk (3 misses)
Performance evaluation: realistic conditions

- The performance analysis has been carried out by using:
  - 7 PCs, each equipped with 2 CPUs Pentium II 350 MHz, hard disk EIDE 4GB, 128 MB of RAM, Fast Ethernet NIC
  - interconnected through a Fast Ethernet switch

- A PC was used as a web switch, while the other 6 computers was used both as
  - servers running the Apache Web server (rel. 1.3)
  - and as clients

- The SURGE Web traffic generator developed at the Boston University was installed on the clients
Performance evaluation: realistic conditions

- SURGE is able to emulate the behavior of a typical Web user by modeling his on/off time activity sequence.

- Differently from other Web benchmarks, SURGE introduces the concept of “User Equivalent” (UE) to capture the notion of throughput.
  - Each UE is modeled as a thread executing the loop `<load(object), wait(off time)>` concurrently with other threads.

- The Web content used consists of distinct files replicated on each real server, where the Zipf distribution of the object’s popularity was generated assuming 20000 requests for the most popular object.
To measure some basic parameters, such as the sustainable throughput and the mean latency, we used SURGE as a simple benchmark, eliminating the off time and using only one UE.

This way, requests are sent in sequence and generate a throughput that is the reciprocal of the latency.

<table>
<thead>
<tr>
<th>Real Servers</th>
<th>Gets/Sec</th>
<th>Total requests</th>
<th>Different requested Objects</th>
<th>Mean size of requested objects [byte]</th>
<th>Mean latency [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS1</td>
<td>91,4166</td>
<td>5485</td>
<td>310</td>
<td>8449</td>
<td>3.321</td>
</tr>
<tr>
<td>WS2</td>
<td>73,1017</td>
<td>4313</td>
<td>331</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS3</td>
<td>136,8999</td>
<td>8214</td>
<td>302</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS1</td>
<td>56,1147</td>
<td>3423</td>
<td>523</td>
<td>8730</td>
<td>5.778</td>
</tr>
<tr>
<td>WS2</td>
<td>56,6949</td>
<td>3345</td>
<td>519</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS3</td>
<td>60,9830</td>
<td>3598</td>
<td>551</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS1</td>
<td>59,5423</td>
<td>3513</td>
<td>518</td>
<td>8756</td>
<td>5.678</td>
</tr>
<tr>
<td>WS2</td>
<td>59,5423</td>
<td>3513</td>
<td>532</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WS3</td>
<td>59,5243</td>
<td>3513</td>
<td>513</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A further analysis was aimed at characterizing the behavior of HRW when the number of servers grows.

For this test, we used a Web content composed of 700 files (medium size = 32 KB, maximum size = 4096 KB) replicated on a varying number of servers (from 1 to 5) and SURGE ran with 100 UEs for 500 secs, with the on/off model activated.
Performance evaluation: worst case

- We chose the configuration characterized by three servers to analyze the behavior of the web switch under **high load conditions**
  - With this configuration HRW shows a worse behavior compared to the other configurations under the same load conditions
  - We performed a number of tests with a varying number of UEs (from 90 to 300)
We analyzed the log files of the real servers with a cache simulator based on the LRU policy.

Using HRW, the fraction of requests that generates hits is higher than those observable with LC and RR.

HRW generates an unbalancing of requests due to the different popularity of web objects belonging to the web site.
A hybrid locality-aware algorithm based on cache prediction

**LACP uses:**
- A content-aware distribution strategy based on a hash function when the cluster load is low
- When the load is high, HRW or LC depending on the value of the load

**LACP uses two thresholds**
- Thigh, Tlow

For each real server, a simulator of the disk cache based the LRU policy is used
Performance improvements

- LACP is able to execute in different configurations:
  - Pure HRW
    - when real server load <= Tlow
  - Pure LC
    - when real server load >= Thigh
  - Pure Cache Prediction scheme (CP)
    - when Tlow < real server load < Thigh

- With 300 Users and three web servers, the latency measurements were those reported in the table below

<table>
<thead>
<tr>
<th></th>
<th>Tlow</th>
<th>Thigh</th>
<th>Algorithm</th>
<th>Latency [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>60000</td>
<td>CP</td>
<td>58,781</td>
</tr>
<tr>
<td>2</td>
<td>60000</td>
<td>60000</td>
<td>HRW</td>
<td>54,965</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td>LC</td>
<td>63,292</td>
</tr>
<tr>
<td>4</td>
<td>4000</td>
<td>4300</td>
<td>LACP</td>
<td>46,106</td>
</tr>
<tr>
<td>5</td>
<td>3500</td>
<td>4000</td>
<td>LACP</td>
<td>48,358</td>
</tr>
<tr>
<td>6</td>
<td>4300</td>
<td>5000</td>
<td>LACP</td>
<td>49,971</td>
</tr>
<tr>
<td>7</td>
<td>4000</td>
<td>4500</td>
<td>LACP</td>
<td>46,875</td>
</tr>
</tbody>
</table>
Conclusions

- A variant of TCP-handoff for the dispatching of HTTP requests in clustered Web servers has been presented.

- The implementation of the mechanism requires a change in the network module of an OS of both the web switch and real servers.

- To evaluate the proposed mechanism, the *Highest Random Weight* name-based scheduling algorithm was implemented and tested.
  - The test bed was composed of a cluster of machines interconnected by a Fast Ethernet network and affected by a HTTP traffic locally generated by the SURGE traffic generator.

- The performance results suggested a further improvement.

- So, a hybrid locality-aware algorithm based on cache prediction was defined and implemented.
  - This algorithm improves the performance of the HRW one.
Future work

We intend to:

- port the TCP-handoff variant implementation on the kernel 2.4.x in which the netfilter framework will be used
- individuate a criterion for defining the $T_{low}$ and $T_{high}$ thresholds
- extend the proposed dispatching mechanism to split of TCP connections over different real servers, necessary to guarantee a higher hit/miss ratio when the HTTP/1.1 is used
- explore specific content-aware algorithms for retrieving dynamic Web pages and multimedia contents