WWW Server Optimizations

John-Patrick Wowra
Email: JohnPatrickWowra@web.de
Table of Contents

• Introduction
• Segment Based Proxy Caching for Distributed Cooperative Media Content Servers
• Performance Issues in WWW Servers
• Conclusion
Introduction
Introduction

• Problem:
  – Extreme Growth of live media services
    • Audio & Video Streaming
  – Limited Capacity of network
  – Significant increases in:
    • Latency
    • Network congestion for Internet Applications
Organization

• Global View:
  – Segment Based Proxy Caching for Distributed Cooperative Media Content Servers
    • Simulation on:
      • Proxy Servers
      • Distributed Cooperative Web Servers (DCWS)

• Detailed View:
  – Performance Issues in WWW Servers
  – Evaluation techniques for improving OS and network protocol software support
    • New Socket Functions
    • Per-byte optimizations
    • Per connection optimizations
First Part

Segment Based Proxy Caching for Distributed Cooperative Media Content Servers
Segment Based Proxy Caching

Figure 1. Flow-diagrams for proxy-based services. (a) The proxy server contains the document, the document is directly sent to the client. (b) The proxy needs to contact the content servers to obtain the document.
Segment Based Proxy Caching

- Latency between proxy & client small in comparison to the latency between proxy & content server
- Proxy Server must contain enough initial blocks to avoid a delay
- Not yet cached media can be pre-fetched
- Storage space needed to store a whole media file too large
- Not a realistic solution for performance improvement
Segment Based Proxy Caching

• Idea:
  – Divide the media files into segments
  – Perform a weighted caching on segments
Segment Based Proxy Caching

- Division of media files
  - Simple segmentation method is used
  - Divides media file into multiple equal sized blocks of transmission units
  - Proxy groups multiple blocks to a segment
  - The size of a segment is sensitive to the distance to the media file
  - Number of blocks in segment n is equal to $2^{(n+1)}$
  - Segment (n) has twice the size of segment (n-1)
Segment Based Proxy Caching

• Cache admission policy/ control
  – Idea: Fill the cache with most frequently used segments from media objects.
  – Different criteria are applied to the prefix part of an object; e.g. the segment number

• Cache replacement policy
  – Least recently used segments are removed from the stack
  – Factors like caching weight of a segment/ distance etc. are used to decide this
Distributed Web Server

• Goal of Distributed Cooperative Web Servers (DCWS):
  – Find application level techniques for distributing web content.
  – Hyperlinks, stored within the documents are manipulated
  – Several conditions and factors shall be considered to determine, which documents should be migrated
  – An algorithm is developed to decide, which documents to migrate
Algorithm for Document Migration

**Algorithm** Document selection for migration

**Input:** Given a local document graph of a home server, and a threshold T of load

**Output:** This algorithm selects a document to be migrated to a cooperative server.

(i) Let the candidate document set C be a set of all documents in the graph.
(ii) Remove all the well-known entry points from C. If C is empty, return nil.
(iii) Remove documents from C if their load is less than the threshold value T. If C is empty, reset it to the previous set and repeat this step with reduced value of T until C becomes nonempty.
(iv) Select documents pointed to by the minimum number of link-from documents that do not reside on the home server.
(v) If two or more documents are selected in step (iv), pick those that point to the minimum number of link-to documents.

**End Algorithm**
Simulation Setup
Segment Based Proxy Caching

Test Results
Test Results (1)

The server throughput values for the DCWS without the proxy server

- 5 servers
- 10 servers

Server throughput (MB/sec) vs. Request rate (number of requests/sec)
Test Results (2)

The server throughput values for 10 and 5 servers after the proxy server is added

- 5 servers
- 10 servers

The server throughput (MB/sec)

Request rate (number of requests/sec)
Test Results(3)

The cache hit ratios for different cache sizes

- 5 servers
- 10 servers

The cache hit ratio

The cache size (number of blocks)
Test Results (4)

The delay ratio on different cache sizes

- 5 servers
- 10 servers

The delay ratio

Cache size (the number of blocks)

Advanced Topics in Computer Networking (SS’05)
Conclusions

• Network performance improves
  – (at least in the simulation)
• Further research is needed to
  – Optimize values, like number of cooperative servers
  – Observe the behavior when the servers have more than one serving thread working concurrently in the system
Second Part

Performance Issues in WWW Servers
Introduction

• Different techniques for improving OS and network protocol software support for high performance WWW servers are evaluated:
  – New Socket functions
  – Per-byte optimizations
  – Per-connection optimizations
New Socket Functions

- Microsoft > Windows NT
  - Acceptex()
  - Transmitfile()

- HP > HP-UX
  - send_file()
HTTP Transaction

<table>
<thead>
<tr>
<th>Step #</th>
<th>Web Server Operation</th>
<th>User-level Optimization</th>
<th>Operating System Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>accept()connection</td>
<td>fast clock reads</td>
<td>acceptex()</td>
</tr>
<tr>
<td>2.</td>
<td>getsockname() for peer</td>
<td>URI cache</td>
<td>acceptex()</td>
</tr>
<tr>
<td>3.</td>
<td>read() request</td>
<td>cache stat() info</td>
<td>acceptex()</td>
</tr>
<tr>
<td>4.</td>
<td>setsockopt() Nagle off</td>
<td>cache mmap()’ed files</td>
<td>inherit socket option</td>
</tr>
<tr>
<td>5.</td>
<td>gettimeofday()</td>
<td>file descriptor cache</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>HTTP request parsing</td>
<td>cache stat() info</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>stat() for requested file</td>
<td>cache mmap()’ed files</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>open() requested file</td>
<td>cache mmap()’ed files</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>read() file into server</td>
<td>writev()</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>write() HTTP header</td>
<td>writev()</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>write() file data</td>
<td>cache mmap()’ed files</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>close() file</td>
<td>file descriptor cache</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>close() socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>write() log entry</td>
<td>buffered write</td>
<td></td>
</tr>
</tbody>
</table>
Per-byte Optimizations

• Data touching operations are expensive
• Unix filesystems force data to be copied when it is moved from one subsystem to another
• An approximate zero-copy integrated I/O architecture is tested to analyse the performance impact
Per Connection Optimizations

- TCP connection was not designed for client-server traffic
- Exchange of more packets than necessary
- How can connection overhead be reduced without violating the TCP protocol?
Per Connection Optimizations

- Sequence of exchanged TCP packets in a HTTP transaction
- 6th packet only carries a FIN bit

1. Client: SYN 0:0(0)
2. Server: SYN 0:0(0) ACK 1
3. Client: ACK 1
4. Client: 1:61(60) ACK 1
5. Server: 1:1159(1158) ACK 61
6. Server: FIN 1159:1159(0) ACK 61
7. Client: ACK 1160
8. Client: FIN 61:61(0) ACK 1160
9. Server: ACK 62

Original TCP packet exchange.
Per Connection Optimizations

- The FIN bit is carried by the 5th packet

1. Client: SYN 0:0(0)
2. Server: SYN 0:0(0) ACK 1
3. Client: ACK 1
4. Client: 1:61(60) ACK 1
5. Server: FIN 1:1159(1158) ACK 61
6. Client: ACK 1160
7. Client: FIN 1160(1160) ACK 1160
8. Server: ACK 62

Piggybacking the FIN.
Per Connection Optimizations

- Further reduction possible
- ACK is delayed

1. Client: SYN 0:0(0)
2. Server: SYN 0:0(0) ACK 1
3. Client: ACK 1
4. Client: 1:61(60) ACK 1
5. Server: FIN 1:1159(1158) ACK 61
6. Client: FIN 61:61(0) ACK 1160
7. Server: ACK 62

Delaying ACK of FIN.
Per Connection Optimizations

- ACK of server's SYN-ACK was removed
- Another 5% performance increase

1. Client: SYN 0:0(0)
2. Server: SYN 0:0(0) ACK 1
3. Client: 1:61(60) ACK 1
4. Server: FIN 1:1159(1158) ACK 61
5. Client: FIN 61:61(0) ACK 1160
6. Server: ACK 62

Delaying ACK of SYN-ACK.
Experimental Setup and Testbed

- Experimental Setup and Testbed
  - 4 IBM 43P RS/6000 Workstations, with one machine as a server
  - Server got three 100 mb/s Ethernet interfaces
  - Each client is connected point full duplex with the server
Client Workload Generator Software

- WebStone and SURGE were used as workload generators
- WebStone is used as a microbenchmark
  - Concurrent requests for the same file
- SURGE is used as a macrobenchmark
  - Concurrent request for a range of files
Performance Issues in WWW Servers

Test Results
Evaluation of Socket Functions

The `acceptex()` system call makes only a small difference of 1% in small transfers.

<table>
<thead>
<tr>
<th>File Size (KB)</th>
<th>Flash Poll</th>
<th>Flash Poll</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1140.11</td>
<td>1151.89</td>
<td>1.03</td>
</tr>
<tr>
<td>2</td>
<td>1059.26</td>
<td>1072.58</td>
<td>1.26</td>
</tr>
<tr>
<td>4</td>
<td>904.15</td>
<td>913.93</td>
<td>1.08</td>
</tr>
<tr>
<td>8</td>
<td>722.22</td>
<td>727.99</td>
<td>0.80</td>
</tr>
<tr>
<td>16</td>
<td>501.92</td>
<td>504.59</td>
<td>0.53</td>
</tr>
<tr>
<td>64</td>
<td>187.72</td>
<td>188.49</td>
<td>0.41</td>
</tr>
<tr>
<td>256</td>
<td>54.36</td>
<td>54.23</td>
<td>-0.24</td>
</tr>
<tr>
<td>1024</td>
<td>13.55</td>
<td>13.56</td>
<td>0.07</td>
</tr>
</tbody>
</table>
Evaluation of Per-Byte Optimizations

• A send_file() implementation combined with an integrated I/O system which does not copy data provides an increase of throughput by up to 85%

• Offloading the checksum to the network device can improve the performance by up to 9%
Evaluation of Per Connection Optimization

- `Send_file()` provides semantic support to enable piggybacking the FIN on the last data segment
- This eliminates one packet
- Throughput for small transfers increases by 7%
- Delaying acknowledgment for the FIN and SYN-ACK packets can eliminate two more packets, increasing performance by additional 11 %
- Total number of packets in small HTTP exchanges was reduced from nine to six, reducing network utilization by up to 20%
Conclusions

• Only little or no increase using acceptex()
• Reducing packet exchanges should help other TCP based applications
• send_file() combined with an integrated I/O system provides better performance
• send_file() is a general function that can also be used by other network servers like NFS, FTP or SMB
Future Work

• Evaluate mechanisms with HTTP 1.1 workloads
  – Per connection optimization will most likely be less significant
  – Per-byte optimization should be more effective than with HTTP 1.0
General Conclusion
Conclusion

By rational analysis and good ideas the performance of the World Wide Web can be improved which is quite necessary because more and more people are causing network congestion in the World Wide Web.
Wake Up, it's Weekend!