

CS290i - Lecture 15

Fast Web Servers

Scalable Internet Services and Systems, Spring 2001

Thorsten von Eicken
 Department of Computer Science
 University of California at Santa Barbara

Flash web server

† “Flash: An efficient and portable Web server”

† V. Pai, P. Druschel, W. Zwaenepoel, @Rice U.

† Contents

- † Server architectures
- † Optimizations
- † Performance eval



Figure 1: Simplified Request Processing Steps

Server Architectures

† **MP: multi-process**

- † Overlaps disk I/O
- † Global optim difficult

† **MT: multi-threaded**

- † Improves MP
- † Not supported in all Oss
- † Less failure isolation

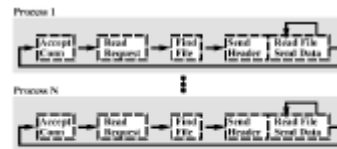


Figure 2: Multi-Process - In the MP model, each server process handles one request at a time. Processes execute the processing stages sequentially.



Figure 3: Multi-Threaded - The MT model uses a single address space with multiple concurrent threads of execution. Each thread handles a request.

Server Architectures

† **SPED: single-process event-driven**

- † Less context switch & memory overhead
- † Use non-blocking I/O
 - † Usually not supported for disk I/O, stat, or open



Figure 4: Single Process Event Driven - The SPED model uses a single process to perform all client processing and disk activity in an event-driven manner.

† **AMPED: asynchronous multi-process event-driven**

- † Uses helper processes
 - † Use mmap to communicate between main and helper
- † Benefits of SPED & MP combined

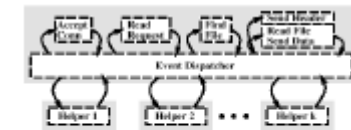


Figure 5: Asymmetric Multi-Process Event Driven - The AMPED model uses a single process for event-driven request processing, but has other helper processes to handle some disk operations.

Optimizations

- † **Pathname translation cache**
 - † E.g. /~tve -> /home/users/tve/public_html/index.html
- † **Response header cache**
 - † Cache HTTPresponse header for each file
 - † invalidate using file cache info
- † **Mapped file cache**
 - † Avoid munmap/mmap sequences
- † **Byte position alignment**
 - † When using writew to combine header & data, avoid odd header lengths

5

Optimizations

- † **Dynamic content generation**
 - † Keep CGI processes alive, e.g. fastCGI
- † **Memory residency testing**
 - † Use mincore to verify mmaped files are in memory

6

Performance

- † **Platform**
 - † P-II 333Mhz, 128MB
 - † >2 100Mbps Ethernet
 - † FreeBSD 2.2.6, Solaris 2.6

7

Synthetic Workload

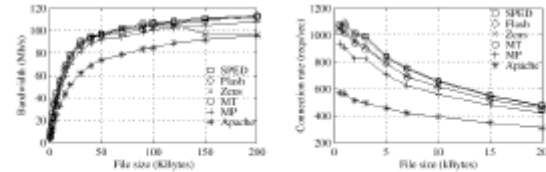


Figure 6: Solaris single file test — On this trivial test, server architecture seems to have little impact on performance. The aggressive optimizations in Flash and Zeus cause them to outperform Apache.

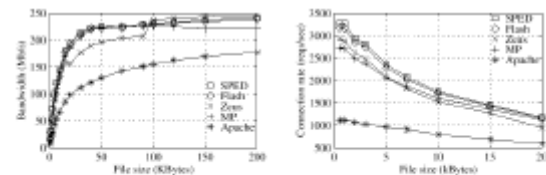


Figure 7: FreeBSD single file test — The higher network performance of FreeBSD magnifies the difference between Apache and the rest when compared to Solaris. The shape of the Zeus curve between 1k kilobytes and 100 kilobytes is likely due to the byte alignment problem mentioned in Section 5.5.

8

Real Workload

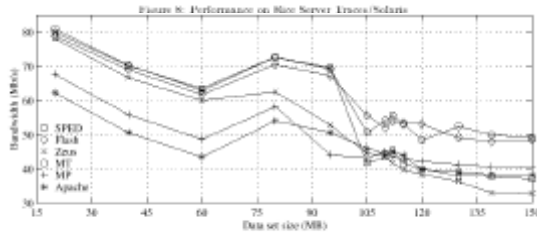
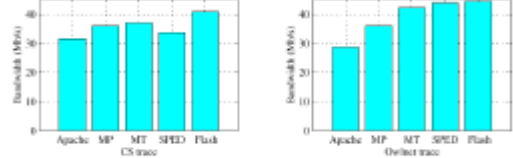


Figure 10: Solaris Real Workload - The Flash-MT server has comparable performance to Flash for both in-core and disk-bound workloads. This result was achieved by carefully minimizing lock contention, adding complexity to the code. Without this effort, the disk-bound results otherwise resembled Flash-SPED.

Summary

- † AMPED combines best of other types
- † Cached workloads: SPED works well, other have more overhead
- † Disk-based workloads: need to overlap disk I/O
- † FreeBSD faster than Solaris
- † AMPED more efficient than MP: needs enough processes to keep disk busy, not one per network connection

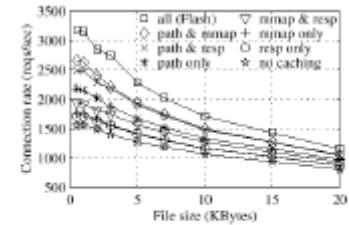


Figure 11: Flash Performance Breakdown - Without optimizations. Flash's small-file performance would drop in half. The eight lines show the effect of various combinations of the caching optimizations.

Distributed Hash

† Scalable Content-aware Request Distribution in Cluster-based Network Servers

† M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, @Rice U.

† "Content-aware Request Distribution"

- † Take into account the content requested, e.g. URL
- † To improve hit rates
- † To allow partitioning
- † To specialize back-end servers
- † Assumptions:
 - † Need software-based solution due to complexity
 - † Need centralized solution

One box

- † ... not enough
- † Splicing - NAT
- † Handoff: forward TCP state

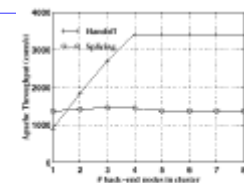


Figure 2: Throughput, 0 KB requests

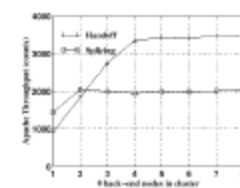


Figure 4: Throughput, IBM trace

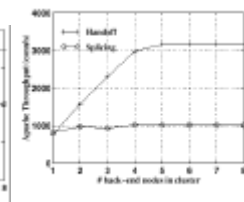


Figure 3: Throughput, 13 KB requests

Scalable design

- † Switch
 - ‡ Round-robin request distribution in HW
- † Dispatcher
 - ‡ Looks at URL
 - ‡ Makes decision
 - ‡ 0/8us/req
- † Distributor
 - ‡ Forwards TCP connections
 - ‡ 300us/req

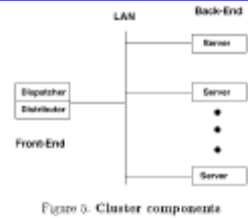


Figure 5: Cluster components

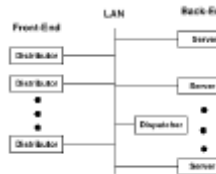


Figure 6: Multiple front-ends



Figure 7: Co-located distributors and servers

13

Synthetic workload

- † Platforms
 - ‡ 300Mhz P-II, 128MB

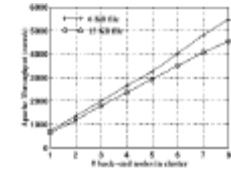


Figure 9: Cluster Throughput

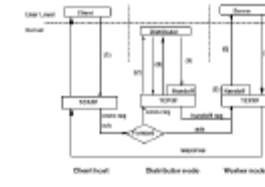


Figure 8: Operation with DNS round-robin

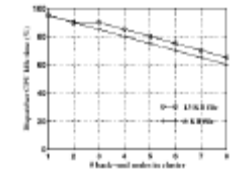


Figure 10: Dispatcher CPU Idle Time

14

Trace-based workload

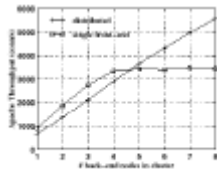


Figure 12: Throughput, IBM trace

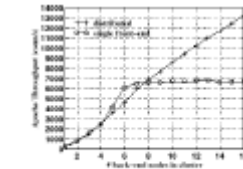


Figure 14: Throughput, Rice trace on 300MHz Athlon

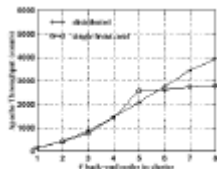


Figure 13: Throughput, Rice Trace

15