Instrumenting and Tuning A Very High Performance Web Server

Douglas L. Urner
Berkeley Software Design, Inc.
<dlu@bsdi.com>
www.bsdi.com

Overview

- What's the real problem?
- Benchmarks for web traffic
- Tuning your web server for high performance
- Tuning your content for 'fast appearance'
- Really big sites
- Futures

What Is a Busy Web Site Today?

- Over one million hits per hour
- T3 – 44.7 million bps
  - Not a skinny little T-1 (@ 1.54 million bps)
  - Multiple T-3s at big sites
- Every bit of performance counts at 40+ million hits/day
- Keys:
  - Efficient software
  - Fast hardware
  - Optimal Configuration

Measuring HTTP Performance

- Important: Web performance → Web usability
- Hype and confusion abound
- Big bucks are at stake
- It's challenging:
  - Lots of variables
  - Only a few in your control
  - Large networks hard (and costly) to simulate
  - Web performance is often very subjective

Qualities of a Good Performance Metric

- Easy to measure in the lab
- Relevant to real world experience
- Comprehensible
- Good metric embraces:
  - Network performance
  - Software performance
  - Hardware performance
  - Client Load

HTTP Benchmarks: WebStone

- First commonly used HTTP benchmark
- Developed by SGI
- Many versions, loosely defined loads and run procedures
- Difficult to compare results
- Can be tailored to your environment
- Available: http://www.sgi.com/Products/WebFORCE/WebStone/

HTTP Benchmarks: SPECweb96

- Commercial “standard” HTTP benchmark
- Developed by many, including: DEC, Hewlett-Packard, IBM, Intel, Netscape, OpenMarket, Silicon Graphics, and Sun
- Not yet widely used
- Well defined loads and run procedures ease comparison of results
- Hard to use as a universal tool for tuning your site because of its model
- Available: http://www.specbench.org/osg/web96/

Typical Benchmark Statistics

- Connections per second: average number of client connections per second accepted by the server
- Latency: average time to complete a request
- Throughput: average aggregate data rate to the net
- Error Rate: typically the rate of "connection refused" errors

Performance Measurement Challenges

- Standardization
  - How does your environment compare to the standard?
  - Can different environments be compared?
- "Metric Inflation"
- Optimization for very special cases that happen to occur in benchmarks

The Basic Plan

1) Derive realistic expectations (e.g., 10 Mbit for ethernet)

2) Investigate actual behavior

3) Identify bottlenecks

4) Fix bottleneck and repeat

Setting Expectations

- Determine overall performance from system components:
  - Theoretical Network Bandwidth
  - Theoretical Disk Performance
  - Relative Processor Speed
  - Learn everything about: hardware, software, protocols, diagnostics, environment, performance claims, etc.
- Sets basic strategy
- Look out for hunches!
- Don't set management or marketing expectations yet!

Expectations, Part 2

- More parameters:
  - TCP/IP protocol stack performance
  - File system software performance
  - HTTP protocol performance
- Sometimes these can measured in isolation
- Gain insight into performance limits

Investigate Actual Behavior

- Plan tests that will enable you to test your performance expectations
- Monitor real statistics while tests are running
  - Observe many parameters
  - Vary load
- vmstat – virtual memory statics
- iostat – device I/O statistics
- netstat – network statistics

Investigation, Part 2

- Look out for surprises! Use your eyes and your ears!
  - Unexpected disk activity (caches aren't working)
  - Low network traffic (why?)
  - Low CPU usage (why?)
  - Anything that's too good to be true
- Good tools:
  - tcpdump
  - ktrace

Identify the Bottleneck

- What factor most limits system performance?
- Use top(1) to identify CPU consumption
- Use profiling to identify hot spots in user code and kernel
- Use perl(1) to simplify and summarize raw data (it can print postscript!)
- Ensure your test configuration can drive the test system to full load

Fix a Bottleneck and Repeat

- Fix one bottleneck at a time!
- Go for "low hanging fruit"
  - Get the system configuration right
  - Get the architecture right
  - Tune the application (user) code
  - Tune the kernel
- Removing one bottleneck creates a new one
- Typical bottleneck removal:
  - Add memory if paging
  - Reduce disk activity: use multiple spindles and/or multiple controllers
  - Reduce network traffic and contention: local DNS cache, split traffic to another net

Case Study

- Early 1996 industry claims:
  - 30 to 60 connections/second were good
  - 10,000 connections/hour caused stress
  - Super Bowl ad for DEC touted 100,000 (or so?) connections per day
- 2/96: BSDI chose high WWW performance as corporate goal
- BSDI used reference platform of:
  - 133 MHz P5
  - 64 MB of RAM
  - Single SCSI disk
  - 100 Mbps Ethernet
  - High performance WWW S/W – no special caching

Real Life: Initial Results

- Server CPU bound
  - 10% user time and 90% system time
  - 3700 open network connections, most in TIME_WAIT.
- First round of tuning:
  - maxusers → 128
  - Confirm hardware configuration ok
  - Gather vmstat and netstat info running benchmark under load
- Results (64 clients):
  - 55 cps (connections/second)
  - 1.2 sec average latency
  - 3.4 Mbps throughput

Real Life: Round Two

- Bottleneck had to be in the kernel
  - We *knew* it was connection lookup searching (kernel)
  - Fixed the searching
- Results with 64 clients:
  - 185 cps
  - 0.3 sec latency
  - 11.4 Mbps
  - 3-4x improvement!
- Observations:
  - Server still CPU bound; 40% user, 60% system
  - 10000 open network connections, most in TIME_WAIT

Real Life: Round Three

- Chose to profile the kernel
  - Two modules (tcp_slowtimeo and tcp_fasttimeo) were candidates for improvement
  - Fixed them
- Results (with 64 clients):
  - 191 cps
  - 0.3 sec latency
  - 11.7 Mbps
  - Little improvement! Oops!
- Server still CPU bound with approximately 60% system time

Real Life: Round Four

- Tuned Apache Web server
  - Turn off DNS lookups (HostnameLookups off)
  - Increased server lifetime (MaxRequestsPerChild 0)
  - Remove checks for .htaccess file (AllowOverride None)
- Results (with 64 clients):
  - 248 cps
  - 0.2 sec latency
  - 15.2 Mbps
  - 30% improvement!
- System time reduced to 40-50%

Real Life: Round Five

- Overall performance is pretty good
- Low end latency high
- Optimized Nagle algorithm for HTTP ("TCP slow start")
- Results (with 64 clients):
  - 255 cps
  - 0.2 sec latency
  - 15.6 Mbps
  - Little improvement (except for low end)

Real Life: Round Six

- Install Squid caching HTTP proxy
- Results (with 64 clients):
  - 326 cps
  - 0.2 sec latency
  - 17.7 Mbps
  - 15-28% improvement
- Reduced context switching I/O for static content

Real Life: Round Seven

- Increased processor to 200 MHz Pentium Pro
- Results (with 16 clients):
  - 544 cps
  - 0.03 sec latency
  - 29.0 Mbps
  - 65% improvement (except latency: 6x)
- Still CPU bound, but latencies are lower and throughput is up
- [Ran out of clients' CPU power]
- Time to stop: 47,000,000 hits/day!

What We Accomplished

- Careful tuning achieved huge HTTP throughput improvement
- TCP speedup summary:
  - Hashed connection lookup
  - More efficient timer management
  - Faster connection establishment
  - Improved packetization
- HTTP server tuning:
  - Disable hostname lookups
  - Eliminate unnecessary forking
  - Eliminate unnecessary directory searches

Lies, Damn Lies, and Benchmarks

- Actual Speed vs. Apparent Speed
  - Actual speed – what we've been talking about; pumping out more bits, faster
  - Apparent Speed – Minimizing the delay (latency) before content starts to appear
- Smart design and coding of content (improves apparent speed)
  - Caches are your friends (improve actual speed)

Designing for Speed

- Help out the browser
- Use dynamic content carefully
- Maximize cachable content
- Be sparing in use of 'cookies'
- Generate useful headers

When One Server Won't Do

- Round Robin DNS
- Clusters
  - BIG/ip
  - HydraWEB
  - LocalDirector

What's Next?

- More and more dynamic content
- Client and server side Java
- WebNFS

Summary

- Take the time to understand the problem (or find somebody who does)
- Test (and measure) methodically
- Design your content for fast delivery
- World class performance with "PC" hardware (using BSDI, of course)

Further Reading

- *Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture*, Aaron Brown and Margo Seltzer. http://www.eecs.harvard.edu/vino/perf/hbench.htm
- The Usenet comp.benchmarks FAQ and more: ftp://rtfm.mit.edu/pub/usenet/comp.benchmarks
- The Bandwidth Conservation Society has good tips: http://www.infohiway.com/faster/