Flash: an efficient and portable web server
Context

- Around 2000, a single application — the web server — became a critical piece of the computing infrastructure.
- Lessons from building a fast web server carry over to other server applications as well.
- Goal of the paper: build the fastest possible web server running on a single node (with potentially many cores).
• Discuss:
  • what you liked about the paper?
  • what you disliked?
  • what did not make sense or what was not clear?
High Level Ideas

• Server performance has several dimensions
  • What are those?
• Lots of different choices on how to express and effect concurrency in a program
• Paper argues that event-driven asynchronous I/O has least overhead and greatest scalability but Unix has poor support
Performance Model

- Need to understand all aspects of delivering a webpage
  - How do TCP/HTTP connections work?
  - What is a web page access look like?
  - What does a disk access mean?
  - How is processor sharing performed using kernel/user-level threads?
  - What is the typical load on the system?
Model of a TCP Connection

- TCP flows provide reliable in-order delivery
- Flow control ensures that there is enough buffer space at the destination
- Congestion control reacts to packet loss
- Slow start allows TCP to probe for available bandwidth starting with a conservative estimate of 1 packet per RTT

- What implications does this have for the design of a web server?
Model of a Web page

- Body of the page is HTML content
- Includes links to embedded images and CSS
- Also includes Javascript that can execute at the client and trigger loads of other types of content
- Embedded HTML in the form of iFrames
- Server side computation in the form of CGI, PHP, etc.
- Potentially a multi-tier architecture
Model of an HTTP Fetch

- Establish TCP connection
- Send HTTP get request
- Server reads requested content from the file system
- Server performs server-side computation
- Server sends data to the client

- What implications does this have for performance? for re-designing HTTP? for the web-server?
HTTP Improvements

- Multiple concurrent connections per client
  - Early browsers: 4 concurrent connections
  - HTTP/1.1 spec: no more than two per hostname
    - browsers ignore this guideline; tend to do ~6 per hostname and subdomains are separate
  - What implications does this have for TCP?

- Persistent HTTP connections
  - Single congestion window is learned for the session; avoid slow start for each
  - Fewer packets, less memory on server side, lower overheads
HTTP Improvements

- Pipelining
  - Send multiple back to back requests on a single persistent connection without waiting for replies
  - Server sends replies in same order as requests
  - Ability to mask the latency of HTTP request/response delay

- SPDY
  - Experimental session protocol
  - Multiplexes many HTTP sessions on a single TCP connection; virtualizes many TCPs on a single TCP
  - Eliminates the “in the same order” limitation of pipelining
Model of a Processor

- Processes incur context switching costs, occupy memory (for stack frames)
- User-level threads implemented within a single process; OS knows only about the process and not the threads inside of it
- Kernel threads implemented as OS visible entities; context switching handled by the kernel

- What are the trade-offs between user-level threads and kernel threads? What about processes and kernel threads?
Model of a Disk

- Disks contain tracks (concentric circles) across multiple surfaces (same track on multiple surfaces form a cylinder)

- Access costs:
  - Seek to the appropriate cylinder
  - Wait for the appropriate segment to rotate underneath the disk head

- Performance governed by mechanics ==> improvements are modest over time
  - single disk read is about a few milliseconds
  - throughput is many tens of mb/s

- What implications does this have for the design of a web server?
Back of the Envelope Calculations

- What would you guess is a typical web page load in terms of latency?
- How would you determine the number of “active” web requests on a server?
- Key distinction: “open loop” vs. “closed loop” systems
Flash Paper

- Four alternatives
  - Multiple process (MP): pool of idle processes
  - Multiple threads (MT): similar, but pool of idle threads
  - Single process Event Driven (ET)
  - This paper: a hybrid

- Discuss:
  - what are the different dimensions by which we should compare them?
Concurrency in a web server

- Why do we want to exploit it?
  - Multi-core: want to be able to exploit multiple CPUs concurrently
  - Multiple disks: want to be able to exploit multiple disk arms concurrently
  - Overcoming latency of networks, flow/congestion control
  - Want to be working on a different request while propagation delay of other requests in flight (or if buffers/windows are full)
OS Issues

- Potentially blocking system calls
  - Some system calls may, in practice, block the calling execution context (kernel thread/process)
  - network receive: caller blocks until data is available
  - network send: caller block until send buffer has space available
  - network accept: caller blocks until new connection arrives

- Potentially high latency system calls: file I/O

- Core issue: some way to either
  - have multiple contexts so that it’s OK if they are blocked
  - prevent blocking (i.e., use non-blocking calls)
AMPED

- **Approach:**
  - Use event driven to process network
  - Use MT or MP to process disk, helper processes, etc.
  - Connect using pipes

- **Benefits:**
  - The thing that is likely to capture the most blocking (networking I/O) is the thing that is lightest-weight
  - Have shared-memory, and single thread tweaking it, so avoid synchronization issues

- **Disadvantages?**
Comparison Metrics

- **Concurrency/utilization:**
  - Not be blocked and utilize all resources efficiently
  - SPED blocks on disk I/O leading to low concurrency (also bad on multi-cores)

- **Overhead**
  - Memory overheads, context switching costs, inter-process communication, etc. SPED is least overhead

- **Coordination**
  - MT/MP models require more effort for application-wide information gathering
  - Application-wide data structures are difficult in MP
Performance Tricks

- Use caches for as many things as possible:
  - name translation caches
  - response header caches
- Maintain memory mapped files and send data directly without requiring copies
- Use writev() and padding to minimize overheads
- Test for memory residency before passing task to helper
- Pre-created CGI helper applications
Evaluations

• What does the paper do well and what does the paper not accomplish in the evaluations?
Cachable Experiments
Real Traces
Control Working Set Size

The graph shows the relationship between bandwidth (Mb/s) and data set size (MB) for different applications: SPED, Flash, Zeus, MT, MP, and Apache. The bandwidth decreases as the data set size increases, with slight variations depending on the application.
WAN Performance

![Graph showing WAN performance with bandwidth (Mb/s) on the y-axis and number of simultaneous clients on the x-axis. Different symbols represent different methods: SPED, Flash, MT, and MP. The graph shows how bandwidth changes as the number of clients increases.](image-url)