accept()able Strategies for Improving Web Server Performance

Brecht, et al (Univ of Waterloo), USENIX04

Context: Performance Measurement, Server Performance

Introduction

Internet server performance continues to be important.

Continue to investigate approaches to improve it under heavy loads.

Examine different connection accept strategies for three different server architectures: kernel-mode TUX server, event-driven μserver, and multi-threaded Knot server.

Look at the rate of accepting new connections and balancing this rate with the need to service existing connections.

Use a SPECweb99-like workload.

Approaches

Much work on operating system support for improved performance.

This work looks at strategies for accepting new connections.

It builds on multi-accept idea proposed by Chandra and Mosberger.

Experiment with balancing the number of new connections to accept with how much processing to give to existing connections.

Accept Details

1. Client establishes a TCP connection with server via 3-way handshake—there is a SYN queue.
2. Kernel adds a socket to accept (or listen) queue.
3. Server calls `accept()`, which removes one socket from the accept queue and returns a `fd` to the socket.
4. Linux has a silent limit of 128 in the accept queue.
5. Want to avoid dropping new connections because of full queues.

Look at improving accept strategy to process more connections.

Web Servers

Start with work on first and extend to the others.

- μserver—built by authors. User-level, event-based server.

  Based on using multi-accept strategy with an *accept-limit* parameter that controls the maximum number of connections to accept each time.

- Knot—multi-threaded Web server. Uses a user-level thread pkg.

  Focus on Knot-C, a thread-per-connection model. Modified so each thread can accept multiple waiting connections if available. Once accepted, all connections are processed before accepting again.

- TUX—event-driven, kernel-mode server for Linux. Affords many I/O advantages. Does not need to cross user/kernel boundary.

  Oscillates between accept-phase and work-phase. Drains accept queue in accept-phase. Handles all items in `work_pending` queue in work-phase. The accept-limit controls max number of connections in accept-phase.

Experimental Methodology

Created two clusters, each with one server and eight clients.

Too many experiments to use a single cluster for each—*surprising there wasn't some comparison done, but then it would be comparing architecture rather than strategy.*

Ensure that all servers can cache the entire workload to avoid differences in cache policies.

Emulated a SPECweb99-like workload using *httperf* web tool capable of generating overload.

Experimental Results

1. μserver. Fig 2 shows replies/sec. In overload situation an accept-limit greater than one shows better performance. Also lower queue drop rates.
2. Knot server. Higher accept-limit does accept more connections and has fewer drops, but does not improve performance—balance wrong in not spending enough time processing.
3. TUX. Get a bit better performance with an accept-limit of one. Some analysis with Eqn. 1 and comparing with μserver.

One-Packet Workload

Motivate that a site like cnn.com used a one-packet response to handle flash crowd.

μserver again shows better performance and queue drop rate with a higher accept-limit.

TUX again shows a bit better performance with an accept-limit of one.

Knot shows best performance with an “intermediate” accept-limit of 50.

More Results

Fig 14 shows key results for the authors that using μserver with a infinite accept-limit can yield similar results as the kernel-level TUX server.

Continue with yet more details on why queue drops occur for TUX and μserver occur.

Fig 17 compares response time of TUX and μserver. Again shows that μserver can provide comparable performance as TUX.

Lots of analysis saying the same.!!

Summary

Show benefit of multi-accept, but primarily for their own μserver—not a lot of improvement for other two server architectures.

Also use results to show comparable performance of kernel-level and μserver approaches.

Reasonable, but not great paper.