teepeedee2: a fast webserver in Lisp

John Fremlin

http://john.freml.in

2009 July 4

Outline

Introduction

This talk is about a web platform I made from scratch as a hobby, called teepeedee2. It runs my blog at http://john.freml.in.

- Please ask questions!
- Sorry my Japanese language skills are terrible.
- But please interrupt and ask questions anyway!
- 疑問が湧いたら途中で、いつでも質問をして下さい。
- むしろ、質問してください。
- 質問は日本語でも結構です。(スタッフができる範囲でフォローします)
- 用中文提問也可以。

teepeedee2

Teepeedee2 is entirely written in Common Lisp. The only external C library dependency is libc.

- Fast HTTP server designed for COMET.
- Fast HTTP client.
- Library for implementing networking protocols with continuation passing style transform.
- Fast HTML/XML output system.
- Domain Specific Language for implementing real-time live updated web applications.
- Blog system with real-time comments.
- An interactive real-time card game.

Real-time interactive setup

- WiFi ESSID teepeedee2
- Go to http://192.168.1.13:3000/chat

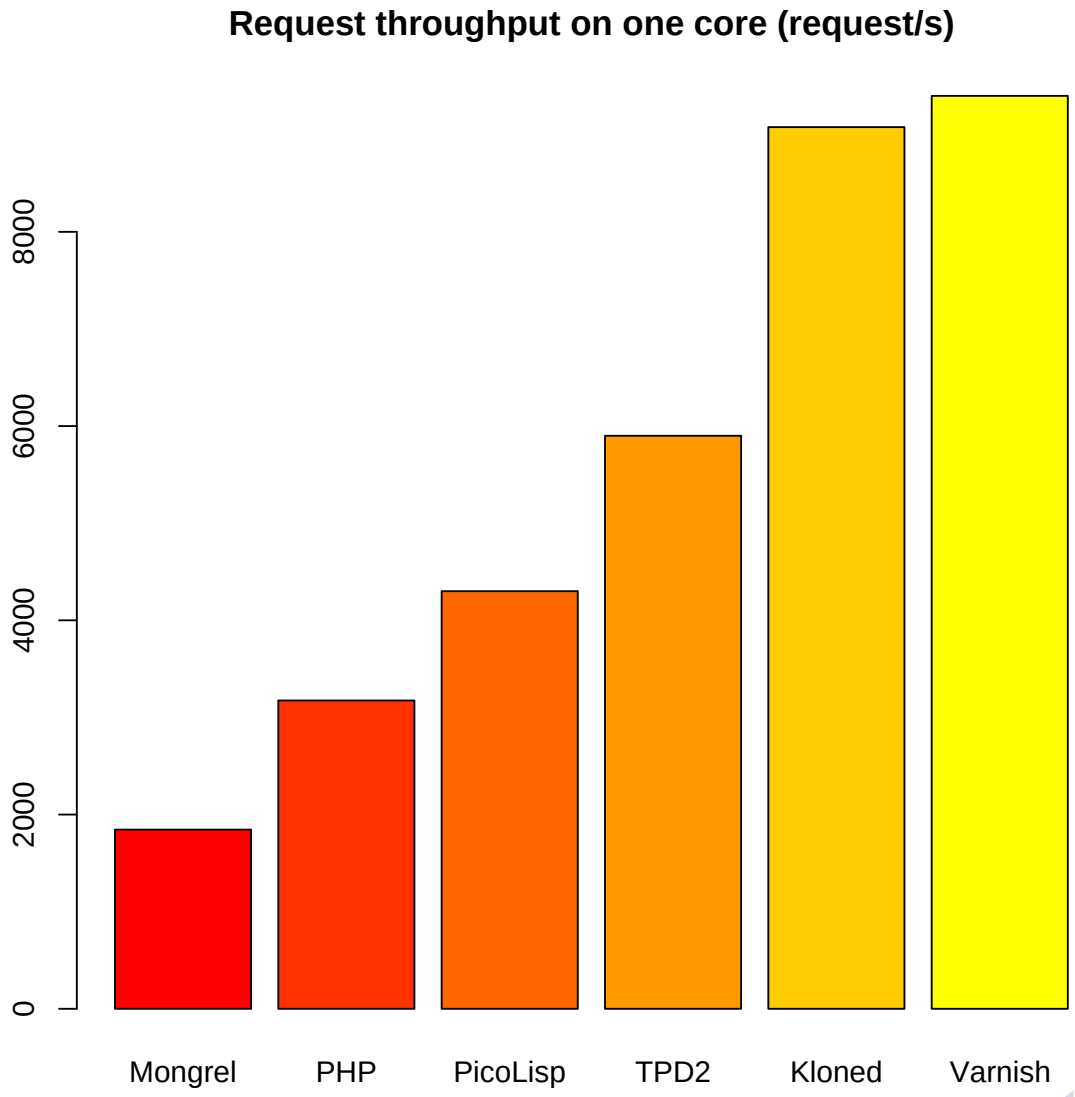
Real-time interactive demo

- Client requests page by HTTP.
- Makes a HTTP request with JavaScript (AJAX) subscribing to different channels - COMET.
- If a channel is modified the server responds at once.
- Otherwise the server waits.
- When a channel is modified all pending connexions have their updates sent.

HTTP server benchmarks

- Measure request throughput for a small dynamic webpage - Hello NAME
- One core only.
- No logging.
- schedtool -a 1 -e ab -n 100000 -c10 http://127.0.0.1...

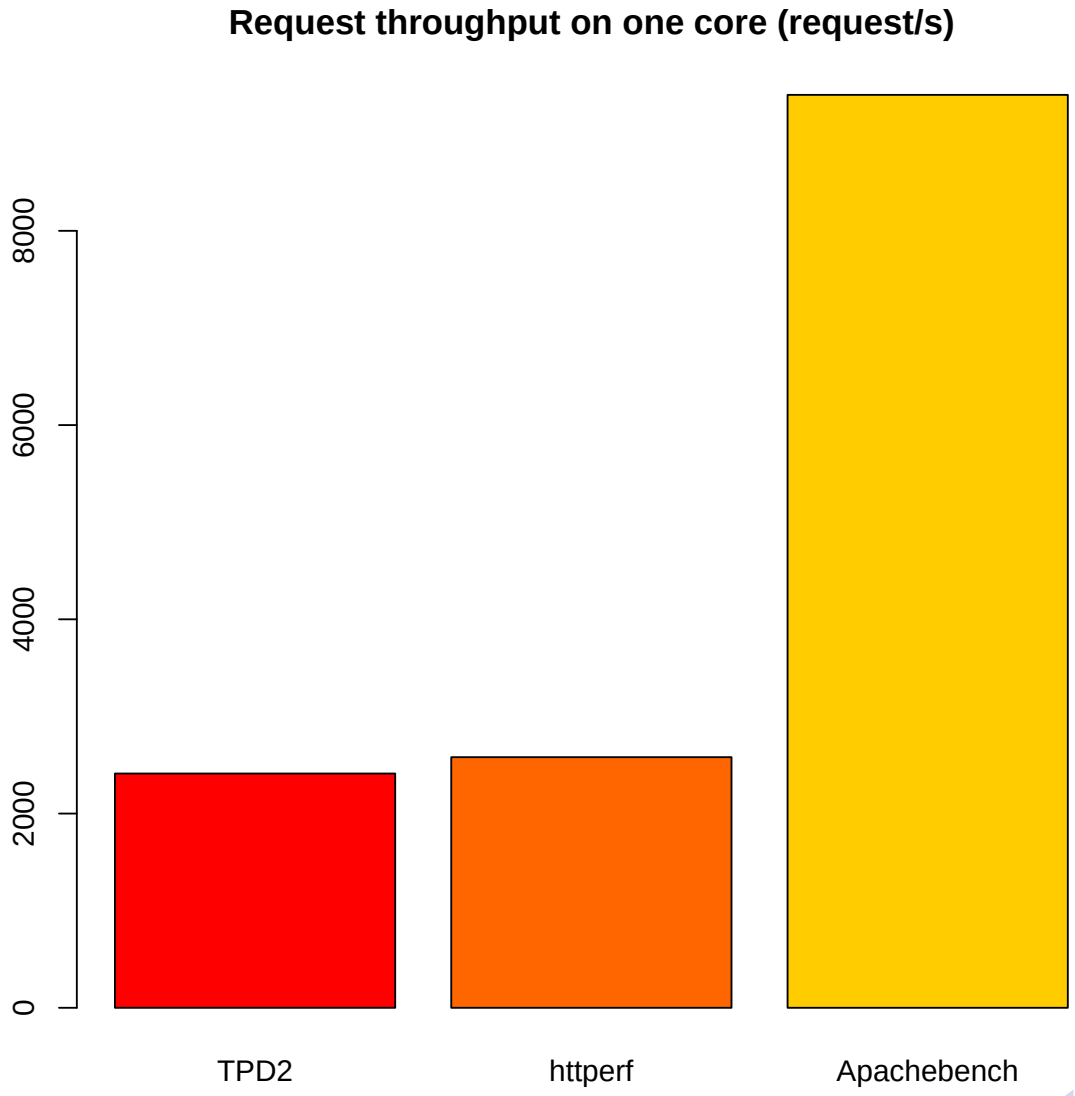
Request throughput



HTTP client benchmarks

- The HTTP client is not tuned.
- Lots of opportunities for improvement.
- But still quite fast - comparable to the httperf benchmarking tool.

Client request throughput



HTTP server under DoS attacks

- slowloris in default mode does not affect tpd2.
- Should be able to scale to 10k connexions without trouble.
- How much more?

Outline

How does it get fast?

- epoll system call (Linux only).
- Consistent use of (unsigned-byte 8) arrays "bytestrings" not encoding to and from Unicode strings.
- Uses cl-irregsexp for fast text processing.
- sendbuf - cords of bytestrings.
- Timeout time complexities are $O(1)$ for insert, delete and $O(m)$ for expiry where m is the number of timeouts to expire - i.e. best possible complexity.
- Not much consing (i.e. little garbage generated).

Outline

my-defun and my

- defstruct creates a lot of long-winded accessors
- In C++, instance slots are already in scope in class methods.
- Let's do the same for Lisp.
- No runtime overhead for "my".
- My first real macro!

my-defun and my example

```
(in-package #:tpd2.user)

(defstruct talk
  (title "Untitled talk")
  (speaker "Anonymous"))

(my-defun talk description ()
   (concatenate 'string (my title)
                " -- "
                (my speaker)))

(my-defun talk 'print-object (stream)
   (print-unreadable-object (talk stream)
     (format stream "description: ~A" (my description))))

;; USER> (make-talk)
;; #<description: Untitled talk -- Anonymous>
;; USER> (talk-description *)
;; "Untitled talk -- Anonymous"
```

defun-consistent

- Precomputes the function if the arguments are known to be constant at compile-time.
- Uses eval if possible.
- Otherwise falls back to load-time-value.
- Cannot use compiler-macros because the output of compiler macros is not compiler macroexpanded on SBCL.

defun-consistent example

```
(in-package #:tpd2.user)

(defun-consistent sorted? (a)
  (prin1 'sorting)
  (equalp a (stable-sort (copy-seq a) #'>)))

;; USER> (sorted? '(9 8 7 6 5 4 3 2 1))
;; SORTING
;; T


;; USER> (defun test () (sorted? '(1 0)))
;; SORTING
;; USER> (test)
;; T
;; USER> (disassemble 'test)
;; ; disassembly for TEST
;; ; 04DDCB54:       BA4F001020       MOV EDX, 537919567
;; ;       59:       488BE5           MOV RSP, RBP
;; ;       5C:       F8               CLC
;; ;       5D:       5D               POP RBP
;; ;       5E:       C3               RET
;; NIL
```

sendbuf

- A list of byte-vectors.
- Saves copying.
- with-sendbuf macrology.
- Big optimisation.

defprotocol

- Uses cl-cont to transform to call/cc continuation passing style.
- Second debug mode that works on normal Lisp streams.
- Would be transparent but need a lot of without-call/cc to stop cl-cont blowing up the compiler.

XML/HTML output

- Spell-checked attributes.
- Parent/child relationships enforced at compile time.
- As much as possible precomputed at compile-time, remainder built up in a sendbuf.
- output-object-to-ml
- DTD definitions.

superquote

- Sometimes ‘(backtick ,comma) syntax cannot be used because it is implemented at read-time.
- Replacement that is expanded as a macro.

CSS output

- Spell checked CSS attributes.
- Shortcuts for some common extensions.
- Allows use of variables in CSS via superquote.

JavaScript output

- Parenscript translates a Lisp-like syntax to pretty JavaScript.
- Pre-compute at compile time if possible.

defpage

Define a web-page and link it to a site.

```
(in-package #:tpd2.user)

(defpage "/add" ((a "1") (b "2"))
  (flet ((i (x) (parse-integer (force-string x))))
   (<h1 a "+" b "=" (+ (i a) (i b)))))

;; USER> (|PAGE-/add| :a 2 :b 4)
;; "<h1>2+4=6</h1>"
;; USER> (type-of *)
;; TEEPEEDEE2.ML::RAW-ML-SENDBUF
```

html-action-form

A higher level construct allowing both the input form and its handler to be specified in the same place.

```
(in-package #:tpd2.user)

(defvar *total* 0)

(defpage "/mul" ()
  (webapp "Calculator"
          (<p "Total is " *total*)
          (html-action-form "Multiply accumulate"
                            ((a 1)
                             (b 1))
                            (flet ((i (x) (parse-integer (force-string x))))
                              (incf *total* (* (i a) (i b))))))))
```

Channels

Browser sessions can subscribe to channels. If the channel is updated, then all the browsers are updated at once.
A strength of tpd2.

- Inherit from the simple-channel class.
- Specialise a method for simple-channel-body-ml.
- Include the channel on a page with output-object-to-ml.

channel example

```
(in-package #:tpd2.user)

(defmyclass (chat (:include list-channel)))
(defvar *chat* (make-chat))

(defstruct message
  text)

(my-defun message 'object-to-ml ()
  (<p (my text)))

(defpage "/chat" ()
    (webapp "Chat"
            (<div (output-object-to-ml *chat*))
            (html-action-form
             "Add a comment"
             ((comment "" :type <textarea))
             (list-channel-add *chat*
                               (make-message :text comment)))))
```

Truc demo

- Highest card wins the hand. (7 is strongest, then 8, A, K, Q, J, 10, 9.)
- Win two hands and you win the pot.
- Increase stake or fold.
- Simple AI with precomputed start cards. Deciding when to raise or fold is hard!

defrules

Define the rules of a game.

- Once again uses cl-cont to transform into CPS.
- Console mode for testing.
- Several general AIs that can play any game.
- Semi-automatically generated web interface.

Outline

Where next?

A unique web platform looking for an application.

This hobby project needs a direction!

Questions and suggestions please.

Where next?

A unique web platform looking for an application.
This hobby project needs a direction!
Questions and suggestions please.

Where next?

A unique web platform looking for an application.
This hobby project needs a direction!
Questions and suggestions please.