He-Man: A domain-specific language for lightweight concurrency

Carlo Angiuli Michael Sullivan

December 12, 2011

1 Introduction

Modern network applications need to support many simultaneous clients—this is known as the C10K ("10,000 client") problem [4]. Modern servers have enough memory and network resources to support this many

To maintain these simultaneous connections, an application must maintain each client's state independently;

while block each one

The simplest way to do this is to spawn a thread for each client, but this approach does not scale as the number of clients grows.

A lightweight approach is to use kernel mechanisms for multiplexed, non-blocking I/O, such as `select`, `poll`, `epoll`, or `kquery`; unfortunately, to maintain separate state for each client, application programmers must manually structure their code as state machines, which is tedious and error-prone. Unsurprisingly, there exist many thin C/C++ wrappers for these facilities, but none avoid major code refactoring.

To maintain the illusion of threads (in particular, client-specific state), languages like Erlang and Haskell provide lightweight threading via user-space schedulers and non-blocking IO. While this affords better performance, the full generality of this approach still imposes unnecessary overhead on networking applications which don't require preemptable, independently scheduled threads [6].

Because

Hence, we intend to produce an embedded domain-specific language in Haskell to describe concurrent network applications. This language will provide composable, high-level network operations, in which programmers may express their application logic naturally, as if using threads.

However, this high-level description will be automatically synthesized into a state machine representation usable in a C loop invoking `epoll`, using no threads whatsoever. We will additionally output a graphical representation of this underlying state machine.

Generate C source code which itself implements the network application, linkable with native C code.

simple runtime

2 Related work

The Nginx [3] and Lighttpd [2] HTTP servers use an event-based architectures, unlike Apache's threads [1], with the specific goal of addressing the C10k problem,

Haskell event loop stuff: [5]
Haskell has epoll support and lightweight threads
Erlang

3 Front-end

4 Back-end

5 Performance

Each thread takes up XXX memory (see Memory.hs)
HTTP server performance is XXX

6 Future work

7 Conclusions

References

[1] Apache MPM worker. `http://httpd.apache.org/docs/2.2/mod/worker.html`, May 2011.

[2] Lighttpd – performance. `http://redmine.lighttpd.net/wiki/1/Docs:Performance`, October 2011.

[3] Nginx wiki. `http://wiki.nginx.org/Main`, October 2011.

[4] Dan Kegel. The C10K problem. `http://www.kegel.com/c10k.html`, September 2006.

[5] Peng Li and Steve Zdancewic. Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '07, pages 189–199, New York, NY, USA, 2007. ACM.

[6] Steve Vinoski. Process bottlenecks within Erlang web applications. *IEEE Internet Computing*, 15:86–89, March 2011.