

Optimising TCP/IP connectivity

An exploratory study in socket-intensive Erlang systems

Bachelor thesis presentation

London, June 14, 2007



Erlang Training and Consulting Ltd
www.erlang-consulting.com

Oscar Hellström
oscar@erlang-consulting.com

Contents

- Erlang based TCP/IP intensive applications
- Stress testing Ejabberd on inexpensive machines
- Surprising results
- Conclusions only a penguin could love

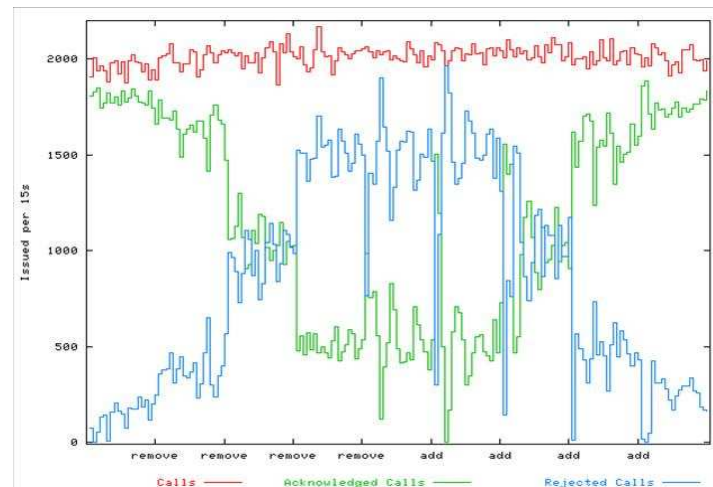
Introduction

- **Erlang TCP/IP intensive applications are common**
 - *HTTP (YAWS, ETC Web Platform)*
 - *Instant Messaging (ejabberd)*
 - *Networking (RabbitMQ)*
- **Scalability of Operating Systems**
 - *Old connectivity benchmarks floating around [1]*
 - *Rapidly changing TCP/IP stacks and OS kernels*
 - *Still not really what this paper is about*

Introduction

- **Scalability of Erlang**

- *Has been investigated in a number of papers [2]*
- *Network scalability has not been addressed*
- *How to combine Erlang applications with a suitable OS?*
- *How to optimise runtime settings of TCP/IP intensive applications?*



Research approach

- **Benchmarking Ejabberd**

- *Tsung, a multi-protocol load testing tool*
- *User scenarios*
 - 20% idle users
 - 60% semi-active users
 - 20% active users

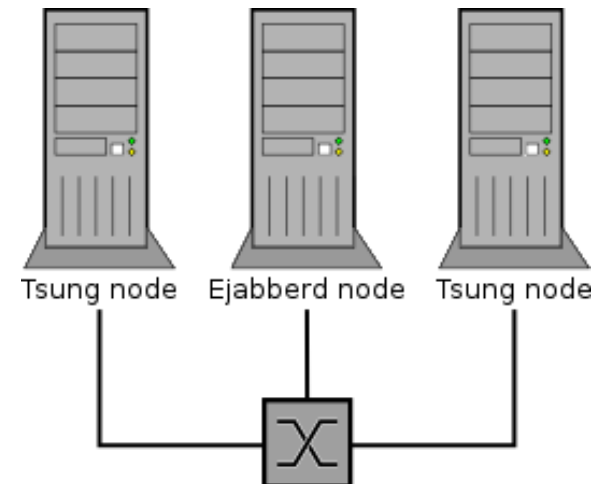
- **Major Operating Systems**

- *Linux (SuSE 9.3)*
- *FreeBSD 6.2*
- *NetBSD 4.1*
- *Solaris 10*
- *Windows port of Erlang does not support kernel poll*



Research approach

- **Low end benchmark machines**
 - *Intel Celeron*
 - *2.4Ghz*
 - *512 MB RAM*
- **2 Tsung machines**
- **1 Ejabberd machine**
- **100 Mbit LAN**



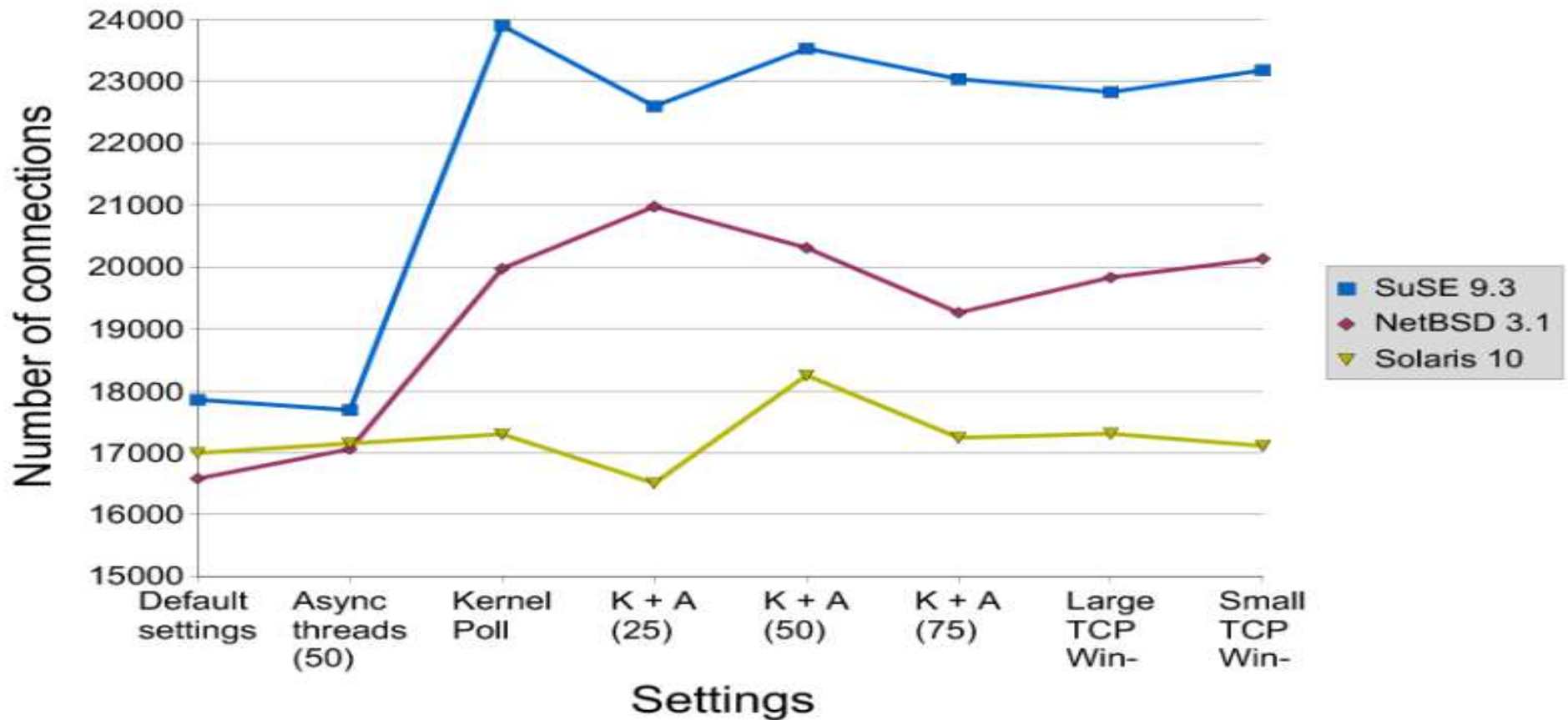
Research approach

- **Erlang Run-Time system settings**
 - *Basic installation*
 - *Asynchronous threads*
 - *Kernel-polling*
- **OS TCP stack optimisations**
 - *Window Size*
 - *Selective Acknowledgments (SACK)*



Results

Concurrently connected users



Results

	Default settings	Async threads (50)	Kernel Poll
SuSE 9.3	17855	17687	23898
NetBSD 3.1	16581	17058	19971
Solaris 10	16994	17150	17300
	K + A (25)	K + A (50)	K + A (75)
SuSE 9.3	22593	23528	23034
NetBSD 3.1	20974	20308	19262
Solaris 10	16506	18248	17241

Results

How did we reach the limits ?

- **Resource limits in the OS**
 - *File descriptors per process*
 - *Sockets per process*
 - *Maximum heap size*
- **Resource limits in the Erlang virtual machine**
 - *Maximum default number of allowed processes*
 - *Maximum number of ETS tables and ports*
- **Hardware constraints**
 - *CPU*
 - *Memory*

Conclusions

- **Operating Systems**
 - *Use SUSE LINUX for small scale systems*
- **Kernel-poll is the must have optimisation**
 - *From a CPU bound system to a memory bound system*
- **Threadpool size is a could have optimisation**
 - *When you need that last bit of performance*
- **OS TCP stack optimisations**
 - *Dependent on the network environment*
 - *Dependent on traffic profile*
 - No impact on reliable IM networks



Contents

- Erlang based TCP/IP intensive applications
- Stress testing Ejabberd on inexpensive machines
- Shocking results
- Conclusions only a penguin could love



Questions



References

1. Felix von Leitner, Benchmarking BSD and Linux,
<http://bulk.fefe.de/scalability/>
2. Nyström J.H. Trinder P.W. King D.J.
High-level Distribution for the Rapid Production of
Robust Telecoms Software: comparing C++ and
Erlang
Concurrency and Computation: Practice &
Experience.
Preprint: <http://www.macs.hw.ac.uk/~trinder/papers/CPE2006.pdf>