Optimising TCP/IP connectivity

An exploratory study in network-intensive Erlang systems

Oscar Hellström

Erlang Training and Consulting Ltd.

oscar@erlang-consulting.com

Abstract

With the increased use of network enabled applications and server hosted software systems, scalability with respect to network connectivity is becoming an increasingly important subject. The programming language Erlang has previously been shown to be a suitable choice for creating highly available, scalable and robust telecoms systems. In this exploratory study we want to investigate how to optimise an Erlang system for maximum TCP/IP connectivity in terms of operating system, tuning of the operating system TCP stack and tuning of the Erlang Runtime System. The study shows how a series of benchmarks are used to evaluate the impact of these factors and how to evaluate the best settings for deploying and configuring an Erlang application. We conclude that the choice of operating system and the use of kernel poll both have a major impact on the scalability of the benchmarked systems.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Measurement techniques; C.4 [*Performance of Systems*]: Reliability, availability, and serviceability

General Terms Measurement, Performance

Keywords Erlang, TCP/IP, Scalability

1. Introduction

The increased use of the Internet together with the increased need for communication between appliances are creating a greater need for network-intensive and concurrent software systems. Many applications developed today are expected to exist in a networked environment, communicating with several other software systems. With this in mind, being able to handle large number of network connections is becoming more and more important.

Erlang is a general-purpose programming language with built-in support for concurrency, distribution and fault tolerance. [14] The language was designed for development of large concurrent soft real-time systems with focus on the telecommunication industry. Previous studies [28] have shown that Erlang is suitable for building robust telecoms systems, due to its ability to achieve high scalability, fault-tolerance and its soft real-time features. Today, Erlang is used for a variety of applications, many of these network-intensive. Examples of such applications are an XMPP/Jabber [31, 32] server [6], an HTTPserver [13] and a web framework [17]. Since network communication usually is dependent on the operating system (OS), it is however not enough to know that Erlang scales well, but the combination of the Erlang Runtime System (ERTS), OS and configurations of the these must also scale well.

Common for most of today's network-enabled applications is the use of TCP or UDP as a transport layer in network communications. The TCP protocol is usually implemented by the OS[1] on which the application is running. It is however possible to replace the OS TCP implementation with alternatives such as a TCP stack implemented in Erlang [29] or one implemented in Standard ML [16]. Apart from reimplementing the TCP/IP stack it is also possible to control how an application is using the OS TCP stack through various configurable options. These settings, together with the

[1] Usually called a TCP stack.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Erlang'07, October 5, 2007, Freiburg, Germany.

Copyright © 2007 ACM 978-1-59593-675-2/07/0010…$5.00

choice of OS will together affect the performance of the software system in respect to maximum number of connections, the total throughput and throughput per socket.

Erlang Training and Consulting develops several applications using Erlang, many of which are both concurrent and network-intensive. This paper aims to use Erlang Training and Consulting in an initial case study for evaluating the best practices for how to configure and optimise the deployment of such a systems to reach maximum scalability in the form of network connectivity.

1.1 Related work

As mentioned above, an implementation of a TCP stack in Erlang exists and is discussed in *A High Performance Erlang Tcp/Ip Stack* [29]. This particular implementation was however designed to achieve fault-tolerance through TCP connection migration rather than performance or scalability. The study however briefly address how scheduling is affected by a large number of sockets, both using the OS native TCP stack and the developed TCP stack.

Liedtke et al. describes how stochastic benchmarking is used in OS research [25]. Even though Erlang is a programming language and not an OS the ERTS has features usually associated with an OS such as processes, scheduling, distribution etc. This paper collects data from benchmarks using a stochastic user simulation model.

Besides the previous mentioned studies, Erlang has also been studied in several other ways, among them in terms of code reuse [26] and performance impacts of specific implementations of certain parts of the standard libraries [19]. This paper wishes to complete these studies by evaluating how existing, network-intensive, Erlang applications are affected in terms of network scalability by different deployment configurations.

2. Research approach

The aim of this paper is to study how different deployment configurations of network-intensive Erlang systems affect connectivity. The goal of these studies are to find best practises for deploying and configuring these systems, with a focus on network scalability. To do this, a number of different deployment environments and configurations will be evaluated through benchmarking. Several parameters will be manipulated between these benchmarks, and the results will be collected and presented through tables and graphs.

This section will describe the approach taken in this study and the main applications and tools used.

2.1 Traffic profiles

When looking at TCP optimisations there are a few different traffic profiles to consider. These traffic profiles have been given four rough characteristics; *short lived* and *long lived* connections and *fast* and *slow* clients. A long lived connection is a client which is maintained for a long period of time while the short lived connection is closed shortly after it was established. A fast client is a client which sends a large amount of data over a short period of time while a slow client will send small amounts of data over a short period of time. An HTTP client would be described as a short lived, potentially fast client while an XMPP/Jabber client usually is a long lived and slow client.

2.2 Test tool

Tsung [5] is a distributed load testing tool which was primarily developed to load test XMPP servers but now also supports several other protocols. Tsung uses a stochastic model for user simulation, with user event distribution based on a *Poisson Process*. This model, which is further discussed in *Traffic Model and Performance Evaluation of Web Servers* [27]. Poisson is known not to be the ideal model for some kinds of TCP traffic [22, 30]. For our purposes of saturating a system it is however sufficient. Users actions are described through different scenarios which specify what the user will do and also how many of the users will choose that particular scenario. The configuration also specifies how often a new user will connect to the service and for how long new users will continue to connect.

2.3 Test subject

Were interested in finding out the maximum number of concurrently connected clients and not interested in network throughput etc. Therefore, the focus of benchmarks will be on slow slow clients that will use long lived connections to the server. Ejabberd [6] is a distributed and fault tolerant Jabber/XMPP server implemented mostly in Erlang. Ejabberd is a network-intensive system, which can be expected to have several thousands of users connected concurrently during normal operation. The connected users are however not expected to generate any large amount of traffic, and

have therefore the characteristics of long lived connections and slow clients.

Ejabberd version 1.1.3 will be deployed using Erlang R11B-4 as main test subject in this study. The ERTS and Ejabberd will be compiled from source and deployed on each OS.

3. Factors

The first factor to be considered is the OS the application is executed on. The other factors likely to have an impact on performance of the application will be evaluated though benchmarks on each operating systems included in this study.

This section will describe the factors that influence the result. Hypotheses concerning the results of altering these factors will also be stated here.

3.1 Factor 1 - Operating system

The choice of OS is very likely to affect the overall performance of the Erlang application in several ways. Benchmarks by von Leitner [21] for example suggest that FreeBSD [8] scales the best in the case of allocating sockets[2] in an application. Von Leitner also shows that Linux 2.6 [9], FreeBSD and NetBSD [10] has the same latency while establishing a large number of connections [3]. These benchmarks are at the time of writing considered old, and addresses outdated versions of the operating systems,[4] even if some remarks about more recent OS versions has been added to the results. Due to the lack of recent data, it is hard to make any hypothesis about the performance benefits of running a particular OS.

The choice of operating systems to evaluate has been based on several factors, such as available commercial support, other benchmarks [21] and current OS knowledge. The most popular implementation of Erlang is distributed as *open source* and supports a wide range of operating systems. Commercial support is however only offered for a subset of these [3]. The benchmark will be performed on the following operating systems:

- SuSE 9.3 [11] (Linux 2.6.11.4)
- NetBSD 3.1
- Solaris 10 [12]

These operating systems are all available as open source.

It was our intention to evaluate FreeBSD 6.2 but the benchmarks could not be completed. The ERTS always terminated with a segmentation fault after some time under heavy load. The reason for these crashes has not been located and further investigations of this would not fit within the scope of this paper.

Not only the choice of operating but also variables such as other running services, file systems used, swap-space and scheduling, which are configurable in most operating systems, will affect the performance of the running application. Therefore minimal installations, where such an option is given by the installer, will be used, and the default configuration of run-levels will be used.

3.2 Factor 2 - Kernel poll

Kernel poll is a name for several techniques which replaces the traditional way[5] of checking for data on a collection of file descriptors with a more efficient and scalable mechanism. The use of kernel poll is likely to affect the performance of the system when a large number, i.e. several thousands, of file descriptors is being used.

Each OS supports at least one different implementation of the technique, even though these are based on the same theory. Linux 2.6 currently implements epoll [24], most BSD variants[6] implements kqueue [23] and Solaris implements `/dev/poll` [2]. There exists patches for the Linux kernel which adds support for Solaris `/dev/poll` interface, but these have been deprecated by epoll. In fact, the `/dev/poll` interface is marked obsolete and has been replaced by *Event Ports* [15] in Solaris 10. The Event Ports are however currently not supported by the ERTS while the `/dev/poll` interface is.

3.2.1 Hypothesis 1

The use of kernel poll will affect the number of concurrent connections positively. This is because the use of kernel poll will decrease the CPU time spent while reading from several thousands of file descriptors.

[2] Executing the standard call `socket()`.

[3] Executing the standard calls `connect()` and `accept()`.

[4] E.g. NetBSD and OpenBSD both implements kqueue in recent versions.

[5] The `poll()` and `select()` calls.

[6] Including recent versions of FreeBSD and NetBSD.

3.3 Factor 3 - Asynchronous threads

Asynchronous threads in the ERTS is a way of stopping blocking calls from interrupting the scheduling of Erlang processes while the calling thread is blocked. The blocking calls are done in separate OS threads, allowing the emulator to schedule the Erlang processes that are not doing the blocking calls. If there is no asynchronous thread available, the job request will be queued until an asynchronous thread becomes idle.

3.3.1 Hypothesis 2

The use of Asynchronous threads will increase the amount of available concurrent connections since the virtual machine will not be blocked waiting for I/O.

3.3.2 Hypothesis 3

The use of too many asynchronous threads will decrease the amount of available concurrent connections since more time will be spent on context switching by the OS.

3.4 Factor 4 - TCP window size

The TCP window size is a way of controlling how many packages that can be put on the network before the sender requires an acknowledgement of the data. Window Scaling, as described in RFC1323, [20] is a way to use larger window size to improve performance on high bandwidth networks. Wechta, Eberlein and Halsall [33] claims that large windows help utilize long, high bandwidth networks, which have a high bandwidth-delay product, but does not help in a local area network (LAN) environment, since there is no need to have many packages in transit. It is possible to set the maximum window size, by means of send and receive buffers for the TCP socket. These values are however also controlled by the application setting up the socket. While the application can specify the preferred buffer sizes when setting up a socket, the minimum and maximum sizes configured by the OS usually cannot be overridden. In some OS it is also possible to configure the TCP stack to always negotiate window scaling, which would allow, but not enforce, large TCP windows.

3.4.1 Hypothesis 4

Window scaling will not affect the number of possible concurrently open connections. XMPP will not create large amounts of traffic and this option will thus not have much impact on the system. Furthermore, the test environment is connected to a small fast and reliable LAN, which has been show not to benefit from large TCP windows [33].

3.4.2 Hypothesis 5

Setting a small window size will would potentially decrease the amount of memory used by the application. Since the application cannot override the maximum buffer set by the OS this will decrease memory consumption, but will not affect the connectivity of the application.

4. Test environment

This section describes the test subject and the configurations together with the configuration of the test tool and environment.

4.1 Test subject configuration

The test subject, see Section 2.3, has been set up on rather low-end hardware. The Ejabberd node is running on a Intel® Celeron® CPU running at 2.40GHz and having 512MB of RAM. The machines are connected to a switched 100Mbit LAN by a SiS900 Fast Ethernet Controller.

4.2 Tsung deployment

The Tsung cluster is running on two machines, identical to the test subject. The two machines are connected to the test subject through a switched LAN and applies load to a single machine, by adding 50 clients per second. This number was the result of initial experiments. At 50 clients per second, all connections are accepted in a timely fashion, before limits are closing up, while the clients does not arrive too slow for the benchmarks to be feasible. Other scenarios, e.g. where new clients are added slower towards the end, was also tested but did not show any impact on the results. Figure 1 shows a diagram over the deployment. The clients will choose different scenarios, where 20 percent of the clients are inactive, sending single chat messages with large gaps, 60 percent are idle, just staying connected receiving presence updates from its contacts and 20 percent are active clients, chatting with other online clients. Figure 2 shows a part of the *inactive* scenario used in the study.

4.3 File systems

The file systems used are the default choice when installing the various operating systems. No tuning of

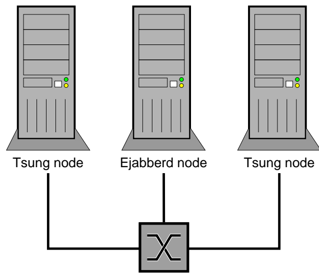

Figure 1. Two test machines applying load to the test subject through a switched LAN.

```
<session name='inactive' probability='20'
    type='ts_jabber'>
  ...
  <request>
    <jabber type="presence:initial"
      ack="no_ack"/>
    </request>
  <transaction name="roster">
    <request>
      <jabber type='iq:roster:get'
        ack="local"/>
    </request>
  </transaction>
  <thinktime value='800'/>
  <transaction name='chat_msg_offline'>
    <request>
      <jabber type="chat" ack="no_ack"
        size="56" destination="offline"/>
    </request>
  </transaction>
  <transaction name='chat_msg_online'>
    <request>
      <jabber type="chat" ack="no_ack"
        size="256" destination="online"/>
    </request>
  </transaction>
  ...
```

Figure 2. Example from inactive Tsung scenario used in this study.

hard drive access has been made after installation. The file systems will affect the test subject's ability to log efficiently. Disk storage is also used for offline message storage.

The file systems used:

- SuSE - ext3 / ReiserFS
- NetBSD - UFS
- Solaris - ZFS

5. Results

This section describes results from the benchmark. The result has been collected by using Tsung's logs. Observations of CPU time and system versus user space CPU, through each operating systems supplied tools, has also been recorded where applicable.

5.1 Common observations

The overall user connection rate are very much the same on all operating systems and settings since this is controlled by the Tsung scenarios. A general connection count curve is shown in Figure 3 and a general error rate curve is shown in Figure 4. The error rate curve plots all error a client can experience throughout a scenario. The curves looks more or less the same for all benchmarks, with the difference being the slope of the curve and the number of time outs error rate.

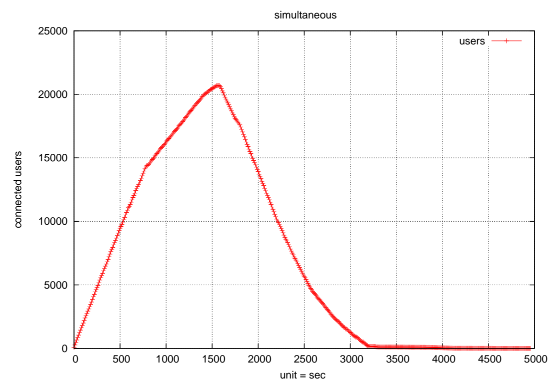

Figure 3. Typical curve for simultaneous connections.

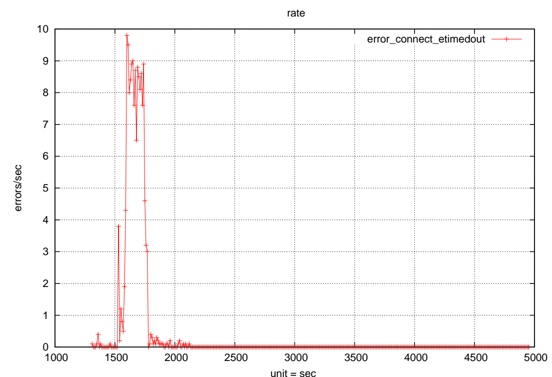

Figure 4. Typical error rate curve showing errors/second.

5.2 Without optimisations

Table 1 shows the number of concurrent connections with the OS and ERTS without optimisations. The ERTS has been given options to allow for more processes, and more *ETS tables*[7] and *ports*[8]. The OS has been configured to allow more file descriptors (and sockets in some cases) than the default settings. These settings aren't really any optimisations, but they need to be changed to allow the Erlang process to allocate as much resources as is needed.

Operating system	Concurrent Connections
SuSE 9.3	17855
NetBSD 3.1	16581
Solaris 10	16994

Table 1. Concurrently connected users without optimisations.

5.3 Kernel poll

Table 2 shows the number of concurrent connections when kernel poll, but no other options, is used.

Operating System	Concurrent Connections
SuSE 9.3	23898
NetBSD 3.1	19971
Solaris 10	17300

Table 2. Concurrently connected users with kernel poll activated.

5.3.1 CPU utilisation with kernel poll

The CPU time is affected by the use of the kernel poll feature. Most notably, on SuSE and NetBSD, system CPU utilisation drops dramatically compared to the benchmarks without optimisations On Solaris, the system CPU drop is not as large, since the system was never CPU-bound on without kernel poll either.

Without kernel poll activated on SuSE and NetBSD, the total[9] CPU utilisation is very close to 100% when the peak of concurrent connections is reached. The high percentage of system CPU leaves very little to user space and it is therefore hard to record if the user space CPU is also affected by kernel poll.

[7] Erlang Term Storage is a built in storage of Erlang terms. For more info see [7].

[8] A port is Erlang's way of communicating with the outside world.

[9] System and user space CPU

	Kernel poll	
OS	**No**	**Yes**
Suse 9.3	82%	3%
NetBSD 3.1	84%	6.5%
Solaris 10	30%	25%

Table 3. Maximum system space CPU usage with and without kernel poll activated.

5.4 Asynchronous threads

Four different settings for asynchronous threads has been evaluated in the benchmarks; 50 threads in the pool without kernel poll and 25, 50 and 75 threads in the pool with kernel poll activated. Table 4 shows the number of concurrent connections with asynchronous threads without kernel poll while Table 5 shows the number of concurrent connections with asynchronous threads and kernel poll.

Operating system	Connections
SuSE 9.3	17687
NetBSD 3.1	17058
Solaris 10	17105

Table 4. Concurrently connected users with 50 asynchronous threads and without Kernel poll.

	Number of threads		
Operating system	**25**	**50**	**75**
SuSE 9.3	22593	23528	23034
NetBSD 3.1	20974	20308	19262
Solaris 10	16506	18248	17241

Table 5. Concurrently connected users with kernel poll and asynchronous threads.

5.5 TCP window size

The benchmark was executed with both a large, 6291456 bytes, and small window size, 8192 bytes. The results of the benchmarks using the different window sizes are shown in Table 6.

5.6 Errors symptoms

From a clients perspective, all system behave very similar when their respective limit is reached for each settings. Connection attempts made to the test subject will time out while no reset connections are noticed.

On SuSE and NetBSD, the test subject behaves almost identical. When the benchmark is run without ker-

	Window size (bytes)	
Operating system	**6291456**	**8192**
SuSE 9.3	22822	23177
NetBSD 3.1	19829	20131
Solaris 10	17307	17109

Table 6. Concurrently connected users with different TCP window sizes in combination with kernel poll.

nel poll, the system is clearly CPU bound, hitting 82% system CPU and 100% total CPU usage. Using kernel poll changes the this, The ERTS process never reaches more than around 50% CPU of which 3% is system CPU, the system is no longer CPU bound. Instead the available RAM is used, and the system starts swapping.

On Solaris, the system is never CPU bound, instead it's always memory bound. The total CPU usage rarely exceeds 65% during the benchmark. Using kernel poll does not change this but it does decrease system CPU slightly.

6. Discussion

We can see that there is a significant difference in the number of concurrent connections in the different benchmarks. There is also a clear difference between the operating systems, where SuSE always performs the best. These results are also visualised in Figure 5. NetBSD and Solaris results are similar without any optimisations, while NetBSD performs better than Solaris when kernel poll is used. The difference in results for the asynchronous threads and TCP window sizes are quite small compared to the increase in performance gained from kernel poll. It could also be argued that these small differences are due to disturbances during the benchmarks.

6.1 Choice of operating system

The results of the benchmarks shows that the choice of operating system has an impact on scalability of network-intensive Erlang system. The best case SuSE benchmark compared to the best case Solaris benchmark shows that SuSE is able to maintain 31% more connections than Solaris. Comparing the best case SuSE to the best case NetBSD shows a difference of 14%. Finally, comparing the best results from NetBSD to the best results from Solaris shows a difference of 15%. The performance of SuSE is highly dependent on the Linux kernel, and it is therefore possible that other distributions of Linux performs equally good.

It is also possible that fine-tuning the operating systems even further could improve the results. It is however hard to predict how significant this improvement would be, as this would be dependent on the researcher's in-depth knowledge of operating systems in general and also the particular operating systems being studied.

6.2 ERTS optimisations

Adjusting ERTS settings produces the same effect on SuSE and NetBSD. The major factor here is the use of kernel poll. On Solaris, this is however not true. Solaris was not CPU-bound without kernel poll activated, which was the case with SuSE and NetBSD. Instead, running Solaris, the system was always memory bound. This indicates that Solaris has a more efficient implementation of the traditional polling mechanism compared to SuSE[10] and NetBSD.

The relatively small impact of kernel poll in Solaris might also be due to the relatively low number of connected users in this benchmark. The performance gain from using an efficient polling mechanism should increase with the number of connections. Also the `/dev/poll` interface has been deprecated in favour of Event Ports [15]. Solaris new Event Ports are planned to be implemented some time in the future [4], which could have an impact on the performance on Solaris.

6.3 Asynchronous threads

The use of asynchronous threads did not show any impact on results of the benchmarks in this study, even though the operations in the test subject are very I/O-intensive. The use of asynchronous threads is however likely to have an impact on performance if a linked in driver would utilise the thread pool more exhaustively [11]. One way to test this would be to use a specialised test subject, implemented to benefit from asynchronous threads.

6.4 TCP optimisations

Tuning of the TCP stack has very little impact on performance for an application where most clients are passive. No exhaustive measurement of network throughput has been documented for this application, but the

[10] Not SuSE specific, but rather Linux specific.

[11] By making many, potentially long, blocking calls.

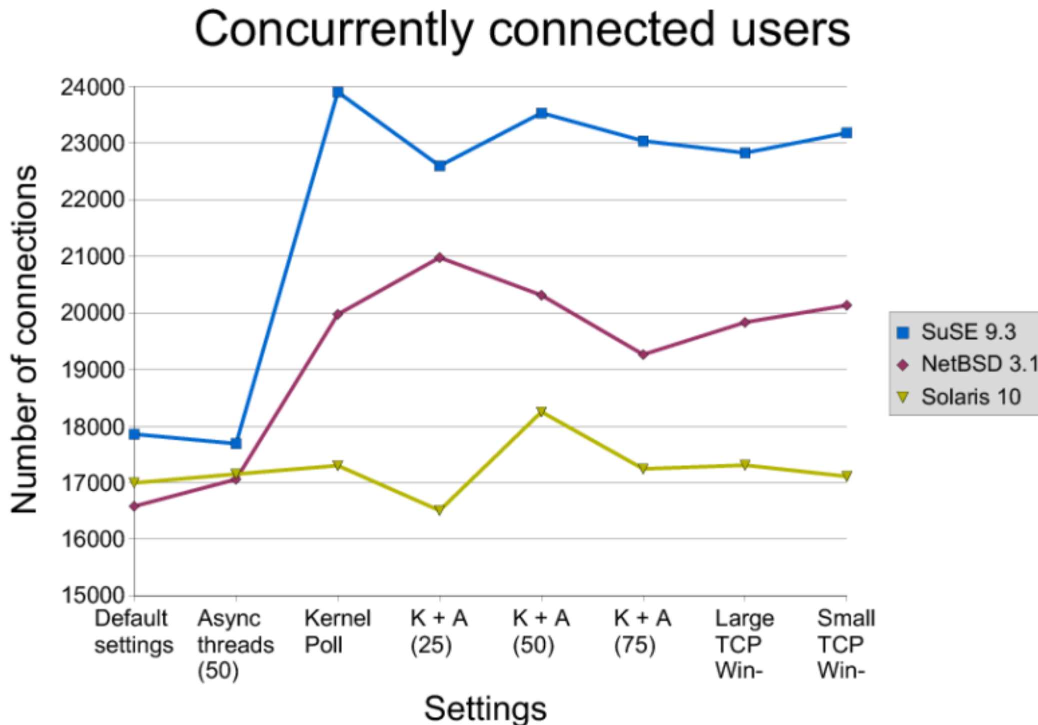

Figure 5. The results collected in one graph. The y axis shows number of connections while the x axis shows the different optimisations.

amount of data sent by the clients is too small to benefit much from these kind of optimisations. Another test subject, i.e. a fast client such as an HTTP or FTP server, is more likely to benefit from this kind of optimisations.

Furthermore, the test environment did not allow for any thorough evaluation of different traffic profiles in combination with network characteristics. To complete these study, deeper investigations network performance and optimisations for certain networks could be done. For instance, doing the same benchmarks on unreliable links could show large differences in impact from any TCP optimisations. This is however not specific for Erlang systems.

6.5 Benchmark environment

It would be impossible to create a completely fair benchmarking environment since certain aspects of the setup will benefit one or the other test subject. Pre-compiled software is optimised for the architecture it was compiled for and the level of optimisation can also vary depending on the complier and the options used. Often pre-compiled OS kernels are optimised for a subset of the instructions available on modern platform, to allow for it to run on a number of platforms. This potentially makes it perform suboptimal on any platform different enough from the one compiled for. These issues are however the same all kernels in these benchmarks, and is part of the reason why custom kernels has been used in all cases.

Not just kernels, but also programs and libraries[12] comes pre-compiled for all systems benchmarked. The way these packages are compiled and linked is known to affect performance. Von Leitner [21] for instance shows how a statical linking can halve the latency in a certain operations.

6.6 Hardware and benchmark results

The fact that the benchmarks were executed on rather low-end hardware is likely to have had an impact on the results. In particular, the improved polling mechanisms are designed to scale very well and there is likely to have been larger differences when polling on more than 20000 file descriptors. Also, when kernel polling is used, the available RAM is exhausted. More memory

[12] This does not include the ERTS or any other part of Erlang which have been compiled on each system, using standard configurations.

would potentially avoid swapping which is very likely to increase performance.

6.6.1 Symmetric multiprocessing support

Symmetric multiprocessing (SMP) support is a relatively new feature in the ERTS which is supported on most of the operating systems Erlang supports. The benchmarks in this paper were performed on a single processor system, without support for hyper threading. With the increased availability of multi-core and hyper-thread enabled CPUs it would also be interesting to benchmark SMP systems.

7. Conclusions

In this study we wanted to evaluate best practices for deploying network-intensive Erlang applications. We have shown that in these benchmarks, the choice of OS and some ERTS configurations have a considerable impact on performance. Considering the results of the benchmarks, we can argue that for low-end systems it is favourable to use SuSE 9.3 to reach maximum scalability. SuSE is at the best able to maintain 14% more connections than NetBSD and 31% more connections than Solaris.

On SuSE and NetBSD the polling mechanism used has a large impact on the CPU usage, while it does not have such large impact on Solaris. The fact that Solaris is not CPU bound without the use of kernel poll indicates that Solaris has a more efficient implementation of the traditional polling mechanisms. Furthermore, this shows that the use of kernel poll has a positive effect for network scalability, but the actual impact is dependent on the particular implementation of kernel poll. Looking at the CPU usage with and without kernel poll, see Section 5.3.1, we can conclude that on SuSE and NetBSD, kernel poll is crucial to the scalability of network-intensive Erlang systems.

Adjusting TCP stack settings does not show any significant impact on the results of the benchmarks in this study. TCP optimisations are likely to make more sense in larger networks, which might have poor links or are crowded, but this falls outside the scope of this study. Performance optimisations in the TCP protocol has been discussed in several other studies, among them *Impact of topology and choice of TCP window size on the performance of switched LANs* [33] and *Performance Impacts of Multi-Scaling in Wide-Area TCP/IP Traffic* [18].

Furthermore, a contribution is made with this paper by examining the validity of the five hypotheses. Kernel poll significantly decreases CPU utilisation in the application due to a scalable way of reading from several thousand file descriptors. Even though the impact is not as large on Solaris, it is still notable. This validates Hypothesis 1, witch was quite expected based on previous studies and reports [23, 1].

The use of asynchronous threads does not increase the number of concurrently connected users notably, which invalidates Hypothesis 2. This is probably a result of the fact that the test subject does not send or receive large amounts of data and is therefore not blocked for any significant period of time while doing I/O. Also, there are probably not enough CPU intensive operations that could be scheduled while waiting for I/O since most operations are not particularly CPU heavy.

Using many asynchronous threads cannot be shown to have an effect on the number of concurrently connected users, which invalidates Hypothesis 3. One likely reason as to why we cannot see any impact of the thread pool is that the asynchronous thread pool in not heavily used by the I/O driver. Therefore all threads are sleeping and no context switching in the OS is necessary.

No impact on performance can be seen in the configuration of TCP window sizes which validates both Hypothesis 4 and 5. The amounts of data sent and received by the clients is likely too small to have an impact on network performance together with the reliable LAN used in the test environment.

8. Acknowledgements

The majority of this work has been carried out during a bachelor thesis project at the IT University of Göteborg, sponsored by Erlang Training and Consulting Ltd. I would like to thank my supervisor Carl-Magnus Olsson for all the help and support during this project.

References

[1] epoll scalability web page. http://lse.sourceforge.net/epoll/index.html, Nov. 2002.

[2] poll(7d) driver for fast poll on many file descriptors. Online manual: http://docs.sun.com/app/docs/doc/816-5177/6mbbc4g9q?a=view, Sept. 2005.

[3] Erlang/OTP product information. http://www.erlang.se/productinfo/index.shtml, Feb. 2006.

[4] Release notes for Erlang R11B-2. http://www.erlang.org/download/otp_src_R11B-2.readme, Nov. 2006.

[5] Tsung. http://tsung.erlang-projects.org/, Dec. 2006.

[6] ejabberd — distributed fault-tolerant Jabber/XMPP server in Erlang. http://ejabberd.jabber.ru/, Feb. 2007.

[7] Erlang/OTP R11B Documentation. http://www.erlang.org/doc/doc-5.5.4/doc/, 2007.

[8] The FreeBSD project. http://www.freebsd.org/, 2007.

[9] The Linux kernel archives. http://www.sun.com/software/solaris/, 2007.

[10] The NetBSD project. http://www.netbsd.org/, 2007.

[11] openSUSE. http://www.opensuse.org/, 2007.

[12] Solaris 10 operating system. http://www.sun.com/software/solaris/, 2007.

[13] Yaws. http://yaws.hyber.org/, 2007.

[14] J. Armstrong. The development of Erlang. In *ICFP '97: Proceedings ofthe second ACM SIGPLAN international conference on Functional programming*, pages 196–203, New York, NY, USA, 1997. ACM Press.

[15] R. Benson. The event completion framework for the Solaris operating system. http://developers.sun.com/solaris/articles/event_completion.html, July 2004.

[16] E. Biagioni. A structured TCP in standard ML. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 36–45, New York, NY, USA, 1994. ACM Press.

[17] F. Cesarini, L. Larsson, and M. Ślaski. From http to html: Erlang/otp experiences in web based service applications. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 52–57, New York, NY, USA, 2006. ACM Press.

[18] A. Erramilli, O. Narayan, A. Neidhardt, and I. Saniee. Performance impacts of multi-scaling in wide-area TCP/IP traffic. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, pages 352–359, Los Alamitos, Mar. 26–30 2000. IEEE.

[19] S. L. Fritchie. A study of erlang ETS table implementations and performance. In B. Däcker and T. Arts, editors, *Proceedings of the 2003 ACM SIGPLAN Workshop on Erlang, Uppsala, Sweden, August 29, 2003*, pages 43–55. ACM, 2003.

[20] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP extensions for high performance, May 1992. Obsoletes RFC1072, RFC1185. Status: PROPOSED STANDARD.

[21] F. v. Leitner. Benchmarking BSD and Linux. http://bulk.fefe.de/scalability/, Oct. 2003.

[22] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.

[23] J. Lemon. Kqueue: Ageneric and scalable event notification system for freebsd. In *Proceedings of the FREENIX Track. 2001 USENIX Annual Technical Conference*, pages 141–153, Berkeley, CA, USA, 2001. USENIX Assoc.

[24] D. Libenzi. /dev/epoll home page. http://www.xmailserver.org/linux-patches/nio-improve.html, 2002.

[25] J. Liedtke, N. Islam, T. Jaeger, V. Panteleenko, and Y. Park. Irreproducible benchmarks might be sometimes useful. In *EW 8: Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 242–246, New York, NY, USA, 1998. ACM Press.

[26] P. Mohagheghi and R. Conradi. An empirical investigation of software reuse benefits in a large telecom producet. Technical report, The Norwegian University of Science and Technology, Computer and Information Science, 2001.

[27] N. Niclausse, Z. Liu, and C. Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Performance Evaluation*, 46(2-3):77–100, Oct. 2001.

[28] J. H. Nyström, P. W. Trinder, and D. J. King. Are highelvel languages suitable for robust telecoms software? In *Proceedings of Computer Safety, Reliability, and Security. 24th International Conference, SAFECOMP 2005.*, pages 275–288, Berlin, Germany, Sept. 2005. Springer-Verlag.

[29] J. París, V. M. Gulías, and A. Valderruten. A high performance erlang Tcp/Ip stack. In K. F. Sagonas and J. Armstrong, editors, *Proceedings of the 2005*

ACM SIGPLAN Workshop on Erlang, Tallinn, Estonia, September 26-28, 2005, pages 52–61. ACM, 2005.

[30] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995.

[31] P. Saint-Andre. RFC 3920: Extensible Message and Presence Protocol (XMPP): Core, Oct. 2004. Status: PROPOSED STANDARD.

[32] P. Saint-Andre. RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, Oct. 2004. Status: PROPOSED STANDARD.

[33] J. Wechta, A. Eberlein, and F. Halsall. Impact of topology and choice of tcp window size on the performance of switched LANs. *Computer Communications*, 22(10):995–965, June 1999.