IJCSIS Vol. 9 No. 1 , January 2011
ISSN 1947-5500

International Journal of Computer Science & Information Security

Ada

Editorial
Message from Managing Editor

The current January 2011 issue of International Journal of Computer Science and Information Security (IJCSIS) continues the tradition of encompassing a wide array of subjects in the fields computer science and information security, ranging from network technologies to security systems. We are very pleased to confirm that IJCSIS is indexed and abstracted in EBSCO and ProQuest, with pending decisions from a number of other database vendors.

Field coverage includes: security infrastructures, network security: Internet security, content protection, cryptography, steganography and formal methods in information security; multimedia systems, software, information systems, intelligent systems, web services, data mining, wireless communication, networking and technologies, innovation technology and management. (See monthly Call for Papers)

We will do our outmost to maintain the established practice of timely publication of regular, as well as special issues of the journal, and further improve the quality of published papers and the journal itself. It will be a great pleasure for us if the readers, reviewers, and authors recognize our intention and efforts, and continue to support us in this mission.

On behalf of the Editorial Board and the IJCSIS members, we would like to express our gratitude to all authors and reviewers for their hard and high-quality work, diligence, and enthusiasm.

Available at http://sites.google.com/site/ijcsis/
IJCSIS Vol. 9, No. 1, January 2011 Edition
ISSN 1947-5500 © IJCSIS, USA.

Abstracts Indexed by (among others):

Google scholar ScientificCommons CiteSeerx beta SCIRUS search engine for science

IJCSIS EDITORIAL BOARD

TABLE OF CONTENTS

Application of Web Server Benchmark using Erlang/OTP R11 and Linux

A. Suhendra[1], A.B. Mutiara[2]
Faculty of Computer Science and Information Technology, Gunadarma University
Jl. Margonda Raya No.100, Depok 16424, Indonesia
[1,2]{adang,amutiara}@staff.gunadarma.ac.id

Abstract— **As the web grows and the amount of traffics on the web server increase, problems related to performance begin to appear. Some of the problems, such as the number of users that can access the server simultaneously, the number of requests that can be handled by the server per second (requests per second) to bandwidth consumption and hardware utilization like memories and CPU. To give better quality of service (*QoS*), web hosting providers and also the system administrators and network administrators who manage the server need a benchmark application to measure the capabilities of their servers. Later, the application intends to work under Linux/Unix – like platforms and built using Erlang/OTP R11 as a concurrent oriented language under Fedora Core Linux 5.0. It is divided into two main parts, the controller section and the launcher section. Controller is the core of the application. It has several duties, such as read the benchmark scenario file, configure the program based on the scenario, initialize the launcher section, gather the benchmark results from local and remote Erlang node where the launcher runs and write them in a log file (later the log file will be used to generate a report page for the sysadmin). Controller also has function as a timer which act as timing for user inters arrival to the server. Launcher generates a number of users based on the scenario, initialize them and start the benchmark by sending requests to the web server. The clients also gather the benchmark result and send them to the controller.**

Key words— Erlang, QoS, Network Management, Concurrent Programming, Distribution

I. Introduction

In the last two decades, human necessities in fast and accurate information create a lot of innovations in information technology, one of them is the internet. Since TCP/IP released to public in 1982 and World Wide Web (WWW) introduced in 1991, internet has became a popular media to access and publish information. The easy to use web mechanisms make people easy to search and publish information on the internet. The web service later grows to many aspects, such as entertainment, education, scientific research and many more.

To access the web on the internet, we need a certain server than can provide user access on the web pages. This server is called web server or HTTP server and has a main duty to serve user access to web pages contents, either static or dynamic.

To give better quality of service (QoS) , web hosting providers and also the system administrators and network administrators who manage the server need a benchmark application to measure the capabilities of their servers. Later, the application intends to work under Linux/Unix -- like platforms and built using Erlang/OTP R11 as a concurrent oriented language under Fedora Core Linux 5.0.

Based on the above descriptions, there are some problems than can be summarized, such as:

1) To give better Quality of Service, web hosting provider and also system administrators and network administrators who manage the web server need a benchmark application to measure the capabilities/ performances of their servers.
2) The benchmark application is intended to be use by the network administrators and system administrators who work under Linux/Unix – like systems.
3) The application is made by utilizing the concurrent capability of Erlang programming language under Linux operating system.

II. Theories

A Web Server

The term web server can mean one of two things [2]:

1) A computer or a number of computers which responsible for accepting HTTP requests from clients, which are known as web browsers, and serving them web pages, either static or dynamic pages.
2) A computer program that provides the functionality described in the first sense of the term.

Web server also works based on several standards, such as [2]: HTTP response to HTTP Request, Logging, Configurability, Authentication, Handling Static and Dynamic Contents, Modular Support, Virtual Hosts

B. Erlang/OTP

Erlang is a concurrent programming language with a functional core. By this we mean that the most important property of the language is that it is concurrent and that secondly, the sequential part of the language is a functional programming language. Concurrent means that the language has focus on how to makes multiple executions threads to run and do computational work together. In Erlang, these execution threads are called processes. The sequential sub-set of the language expresses what happens from the point it time where a process receives a message to the point in time when it emits a message.

The early version of Erlang was developed by Ericsson Computer Science Laboratory in 1985. During that time, Ericsson couldn't find an appropriate language that has high performance in concurrency especially for telecommunication applications programming (for switching, trunking, etc), so they developed their own language. OTP stands for Open Telecom Platform, OTP was developed by Ericsson Telecom AB for programming next generation switches and many Ericsson products are based on OTP. OTP includes the entire Erlang development system together with a set of libraries written in Erlang and other languages. OTP was originally designed for writing telecoms application but has proved equally useful for a wide range of non-telecom that have concurrent, distributed, and also fault tolerant applications. In 1998 Ericsson released Erlang and the OTP libraries as open source. Now, Erlang/OTP has reached the R11 version.

III. Why We Use Erlang?

The simple answer to the question above that is we need concurrency in the benchmark application. The application must be able to generate multiple users to do some stress tests to the web server.

But, there are some good features in Erlang, and even the other languages don't have these features. Some of these Erlang features are described below:

1) In Erlang processes are light weight.
2) Not only are Erlang processes light-weight, but also we can create many hundreds of thousands of such processes without noticeably degrading the performance of the system (unless of course they are all doing something at the same time)[5].
3) In Erlang, processes share no data and the only way in which they can exchange data is by explicit message passing. "dangling" pointers are very difficult to program in the presence of hardware failures - we took the easy way out, by disallowing all such data structures [4].
4) In Erlang, processes scheduling operation is done by its own virtual machine, so Erlang didn't inherit the underlying operating system processes scheduling. Real time. Erlang is intended for programming soft real-time systems where response times in the order of milliseconds are required.[4]
5) Continuous operation.
6) Automatic Memory management.
7) Distribution.

A. *Concurrent and Distributed Erlang*

Concurrent in Erlang involves processes creation and deletion. In order to create a new process in Erlang, we use BIF (Built In Function) *spawn/3* :

```
spawn(modulename,functionname,
argumentlists)
```

or

```
pid_variabe=spawn(modulename,functionna
me, argumentlists)
```

The illustration of process creation can be seen in figure 1.

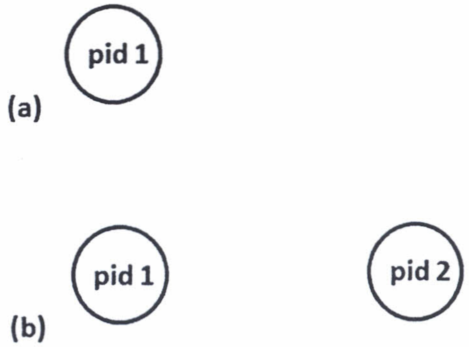
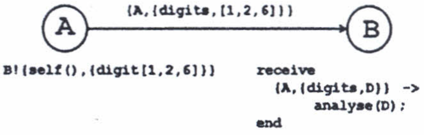


Figure 1. Process Creation Illustration

A process which no longer need by the system will be automatically shutdown/delete by the virtual machine (Erlang Runtime System/ERTS). Meanwhile, the message parsing mechanism can be done by these codes :

```
Pid ! Message
.......
Receive
Message1 ->
Actions1;
Message2 ->
Actions2
.....
After Time - >
TimeOutActions
end
```

The illustration for message parsing can be seen in figure 2.



```
B!{self(),{digit[1,2,6]}}
```

```
receive
  {A,{digits,D}} ->
      analyse(D);
end
```

Figure 2. Message Parsing Illustration

Besides the example that has been shown above, in Erlang we can also use Behaviour in OTP Design Principles to create, delete and do message parsing between processes. A distributed Erlang system is a number of Erlang Runtime System (we called them nodes) that communicated each other by using message parsing with pid (process indentifier) through TCP/IP sockets transparently. A node must be given a name before it can communicate each other. The name is either a long name or a short name like the examples below:

```
$erl -name dilbert[long name]
(dilbert@uab.ericsson.se)1>
@erl-sname dilbert[short
name](dilbert@uab)1>
```

A simple distribution in Erlang can be done by these codes:

```
...
Pid = spawn(Fun@Node)
...
alive(Node)
...
not_alive(Node)
```

B. Benchmark

The term benchmark can be described as:

1) A group of parameters in which products (software or hardware) can be measured the performance according to these parameters.
2) A computer program designed to measure the performance of software or hardware according to certain parameters.
3) A group of performance criteria that must be complied by software or hardware.

Web server benchmark means that a benchmark activity is made to the web server to measure its performance based on several parameters and using certain computer program to do this activity.

IV. Designing

A. Application Concepts

Application is divided into two main parts, the controller section and the launcher section.

Controller is the core of the application. It has several duties, such as read the benchmark scenario file, configure the program based on the scenario, initialize the launcher section, gather the benchmark results from local and remote Erlang node where the launcher runs and write them in a log file (later the log file will be used to generate a report page for the sysadmin). Controller also has function as a timer which act as timing for user inters arrival to the server.

Launcher generates a number of user based on the scenario, initialize them and start the benchmark by sending requests to the web server. The clients also gather the benchmark result and send them to the controller.The illustration for Application Concepts can be seen in figure 3.

Several parameters that can be measured by this application are:

1) Number of Requests per second.
2) Simultaneous users that can be served by the server (per second).
3) Time that needs to connect for the client so it can be connected to server.
4) Number of user that can be served during the duration of benchmark.
5) Time that needs for a user so it can receive a full page of document/web page according to the request.
6) Time that needs to complete a session (a group of requests) as described in the scenario.
7) Network throughput.
8) HTTP Status (200, 404).

The application also has a report generator that written in PERL and using GNUPLOT to generate graphs based on the benchmark results from the log file.

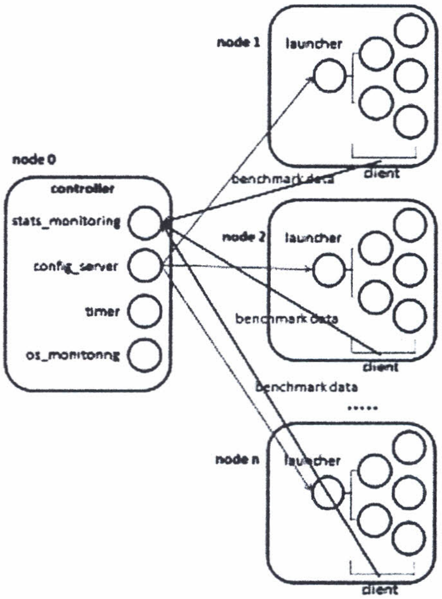


Figure 3. Illustration for Application Concepts

B. The Scenario Files

The benchmark scenario file is written using XML and consists of several sections:

1) The server section, where the user (sysadmin) describe the IP address of the server that he/she wants to benchmark.
2) The client section, in this section, user can write the IP addresses where the launcher section starts and generate a numbers of clients.
3) Inters arrival phase and benchmark duration section.
4) The simulated user agents (web browser) section.
5) The session and request section.
6) Each of the section describes above can be modified according to user necessity.

V. Implementation

A. Hardware and Software Specifications

Application is implemented by using a simple topology consists of two computers and a server across Local Area Network in Kapuk Valley, Margonda, Depok. The two computers using an Intel Celeron Processor (1.8 GHz and 2.28 GHz) and running Linux operating System (SuSE 10.0 and Slackware Linux 11.0), Open SSH v2, Erlang/OTP R11, PERL v5.8, and also BASH Shell. Both of them also connected to the TCP/IP network and within Kapuk-Valley domain (hostname mobile and posen-asik). The server using an Intel Pentium II 400 MHz. processor with 768 megabytes SDRAM memories and running Slackware Linux 11.0 with Apache 2.0.55 web server.

B. Testing and Implementation Process

In this implementation process, we're going to make a benchmark scenario with these parameters:

1) Total clients to generate: 600 clients.
2) Benchmark duration: 10 minutes.
3) Client inters arrival phase: 1 second.
4) 300 clients will be generating in host mobile, meanwhile the other 300 clients will be generate in host posen-asik.

Before we run the program, we must generate a pair of authentication key (public and private) for passwordless authentication using SHH in order to get the Erlang nodes to communicate each other.

```
[root@mobile~]#ssh-keygen -t rsa

Enter file in which to save the key
(/root/.ssh/id_rsa):

Enter passphrase (empty for no
passphrase):

Enter same passphrase again:

Your identification has been saved in
/root/.ssh/id_rsa.

Your public key has been saved in
/root/.ssh/id_rsa.pub.

The key fingerprint is:
ec:30:2c:c9:e0:0a:92:48:3c:e5:5a:f3:7c:69:
d8:92 root@mobile.myownlinux.org

[root@mobile~]#scp /root/.ssh/id_rsa.pub
root@posen-asik:/root/.ssh/

[root@posen-asik]#echo .ssh/id_rsa.pub >>
.ssh/authorized_keys}
```

After the process above, we can start the benchmark process by executing the shell script to initialize and start the application.

```
[root@mobile~]#/usr/local/bin/wii start
```

Several results from the important parameters to examine by the sysadmin are listed in the table I.

TABLE I

BENCMARK RESULTS

Parameters	Result
Request per Second	3.4 requests per second
Connection per Second	1.8 connections per second
Page loaded per second	1.8 pages per second
Total user served	595 of 600 users

VI. CONCLUSIONS

After our studies we show that in Erlang:

1) Processes are light weight. Not only are Erlang processes light-weight, but also we can create many hundreds of thousands of such processes without noticeably degrading the performance of the system (unless of course they are all doing something at the same time),
2) Processes share no data and the only way in which they can exchange data is by explicit message passing. Erlang message never contain pointers to data and since there is no concept of shared data, each process must work on a copy of the data that it needs. All synchronization is performed by exchanging messages.
3) Processes scheduling operation is done by its own virtual machine.
4) Processes are real time. Erlang is intended for programming soft real-time systems where response times in the order of milliseconds are required.
5) Processes code has continuous operation. Erlang has primitives which allow code to be replaced in a running system and allow old and new versions of code to execute at the same time.
6) Processes operation use automatic memory management. Memory is allocated automatically when required, and deallocated when no longer used.
7) All interaction between processes is by asynchronous message passing. Distributed systems can easily be built.

By examine the results based on the important parameters; we hope that the syadmin and netadmin can make fine tuning to their server.

ACKNOWLEDGMENT

The authors would like to thank Gunadarma University's Research Council

REFERENCES

[1] Anonymous, *Panduan Lengkap Pengembangan Jaringan Linux*, Penerbit Andi, Yogyakarta.

[2] Anonymous, *Web Server*, http://en.wikipedia.org/wiki/Web server (05 September 2010).

[3] Anonymous, *An Erlang Course,* http://www.erlang.org/download/course.pdf, (05 September 2010).

[4] Armstrong, Joe. et al., *Concurrent Programming in Erlang*, Prentice Hall, New Jersey.

[5] Armstrong, Joe, Concurrency Oriented Programming in Erlang, Swedish Institute of Computer Science.

[6] Cisco System, Cisco Networking Academy Program – CCNA Modules – 640-801, http://cisco.netacad.net, Washington.

[7] Dudy Rudianto, *Desain dan Implementasi Sistem Operasi Linux*, Elex Media Komputindo, Jakarta, 2002.

[8] Dudy Rudianto, *PERL Untuk Pemula*, Elex Media Komputindo, Jakarta.

[9] Erricson Tellecomunications Systems Laboratories Team, *Erlang Reference Manual*, Erricsson Inc., Sweden.

[10] Erricson Tellecomunications Systems Laboratories Team, *Getting Started With Erlang*, Erricsson Inc., Sweden.

[11] Hedqvist, Pedka, A Parallel and Multhitreaded Erlang Implementation, Uppsala University, Sweden.

AUTHORS PROFILE

A. Suhendra was born in Jakarta, 1969. He received the B.Sc. degree in Informatics Engineering from Gunadarma University, Depok, in 1992, and his MSc in Computer Science from AIT, Bangkok in 1994, and his Dr.-Ing. in Informatics from University of Kassel, Germany in 2002.

A.B. Mutiara was born in Jakarta, in 1967. He received the B.Sc. degree in Informatics Engineering from Gunadarma University, Depok Indonesia,in 1991, and the M.S. degree and Ph.D degree in computational material physics from GoettingenUniversity, Germany, in 1997 and 2000.

ISSN 1947 5500