Erlang/OTP

VOLUME I
A Concurrent World

MANUEL ÁNGEL RUBIO JIMÉNEZ

Erlang/OTP

Volume I: A Concurrent World

Manuel Angel Rubio Jiménez

Translated by
Ana María Rubio Jiménez

Reviewed by
Ayanda Dube

Erlang/OTP
Volume I: A Concurrent World

Manuel Angel Rubio Jiménez

Translated by
Ana María Rubio Jiménez

Reviewed by
Ayanda Dube

Abstract

The Erlang programming language was born around the year 1986 in Ericsson laboratories by the hand of Joe Armstrong. It is a functional language based on Prolog, fault-tolerant, and oriented to real-time work and concurrency, which provides certain advantages in terms of algorithm declaration.

Like most functional languages, Erlang requires an analysis of the problem and a way to design the solution differently than it would be done in an imperative programming language. It suggests a better and more efficient way to carry it out. It is based on a syntax that is more mathematical than programmatic, so it tends more to solve problems than to order and execute orders.

All this makes Erlang a very appropriate language for the programming of critical mission elements, both at the server level and at the desktop level, and even for the development of embedded systems.

This book contains a compendium of information about what language is, how it covers the needs for which it was created, how to get the most out of its way of performing tasks and its orientation to the audience. It is a review from the beginning about how to program in a functional and concurrent way in a distributed and fault tolerant environment.

ISBN 978-84-945523-7-3

[1] http://erlang-otp.es/
[2] http://creativecommons.org/licenses/by-nc-sa/3.0/

Chapter1.What you should know about Erlang

Software for a concurrent world.
—Joe Armstrong

Erlang is becoming an environment and a fashionable language. The growing existence of companies oriented to the provision of Internet services with a high volume of transactions (such as game networking or mobile messaging and chat systems) means that jobs like these are proliferating in different countries such as the United States, the United Kingdom and Sweden, where professionals in this language are required. There is an imperative need to develop environments with the characteristics of the Erlang machine and the development methodology provided by OTP.

In this chapter we introduce the concept of Erlang and OTP: its meaning, characteristics and history. The information in this first chapter is complimented by the sources that have motivated it and provides accurate information on where each section has been extracted.

1.What is Erlang?

To understand what Erlang is, we must comprehend that it is a complete development platform or environment. Erlang not only provides a compiler for the source code, but also has a collection of tools and a virtual machine (called BEAM) to run compiled code. Therefore, Erlang may be viewed from two perspectives:

Erlang as a language

There are many discussions on whether Erlang is a functional programming language or not. On inception, it is understood that it is, although as you progress with it, you notice some elements that make it deviate from this pure classification. Therefore, Erlang could better be classified as a ***hybrid*** language, having elements of a functional type, an imperative type, and even some features that allow some orientation to objects, although not entirely complete.

The best classification of Erlang, at least from my point of view is saying it is a language oriented to concurrency. Erlang has some great native features tailored for distributed, parallel (concurrent) programming as well as inherent mechanisms for ensuring fault tolerance. It was designed from the beginning to be executed in an uninterrupted manner. This means you can change the code of your

applications without actually stopping or interrupting its execution. Later we will explain more concretely how all of this works.

BEAM or Erlang as execution environment

As we have already mentioned, Erlang is a development platform that provides not only a compiler, but also a virtual machine for execution. Unlike other interpreted languages such as Python, Perl, PHP or Ruby, Erlang is compiled and its virtual machine provides an important layer of abstraction that gives it the ability to manage and distribute processes between nodes in a completely transparent manner (without the use of specific libraries). The term node is commonly used to refer to a named instance of the Erlang virtual machine. More on this later.

The virtual machine (or node) on which Erlang's compiled code is executed on, providing all the characteristics of distribution and communication of processes, is also a machine that interprets machine code[1] which, actually has nothing to do with the Erlang language itself at that level. This has allowed a proliferation of languages that use the Erlang virtual machine but not the language itself, such as: Reia, Elixir, Efene, Joxa, Alpaca or LFE.

Erlang was the proprietary code until 1998, when it was ceded as open source to the community. It was created initially by Ericsson, more specifically by Joe Armstrong, although not only him alone, with significant involvements of Robert Virding and Mike Williams.

It was given the name Agnus Kraup Erlang. Although it is also assumed that its name is an abbreviation of ERicsson LANGuage, from its intensive use in Ericsson. According to Bjarne Da#cker, head of the Computer Science Lab at the time, this duality is intentional.

2.Erlang Features

During the time when Joe Armstrong and his colleagues were in the laboratories of Ericsson, they saw that the development of applications based on PLEX was not optimal at all, for programming applications within the hardware systems of Ericsson. For this reason, they began to search for what could have been an optimal development system, based on the following specifications:

Distributed

The system had to be distributed in order to balance out its load across hardware systems. They were looking for a system that could

[1]Or native code chunks if we use HiPE.

launch multiple processes not only on the machine on which it executed on, but also on other peer machines within its network. Similar to what in languages like C is provided for by PVM or MPICH but without the explicit use of any library.

Fault-tolerant

If part of the system had failures and had also to be stopped, it should not have resulted in the entire system being consequently stopped as well. In software systems such as PLEX or C, a failure in the code determines a complete interruption of the program with all its threads and processes. There are other languages such as Java, Python or Ruby that handle these errors as exceptions, affecting only part of the program and not all of its associated threads. However, in shared memory environments, an error can leave memory corrupt, potentially affecting other parts of the program which reference the same memory locations, hence for such reasons, this option did not provide the required system-wide safety guarantees.

Scalable

Conventional operating systems had problems in maintaining a high number of running processes. The telephony systems developed by Ericsson were (and some, still are) based on having a process for each incoming call, which controlled the stages of the call and could trigger events as well as forward them to specific handlers, which in turn would perform the necessary actions using its own processes. Therefore, we were looking for a system that could manage from hundreds of thousands, to millions of processes.

Hot Code Swap

It is very important in the live system environments at Ericsson, and in most critical systems in production of any kind, that they never stop, even if updates have to be made. Maintaining such high uptime of the system despite making live updates, is very important for both the technical and business units. For this reason, Hot Code Swap was also added as a feature to Erlang, allowing code and patch updates to be carried out without necessarily having to stop the system, and without affecting the rest of the executing code.

Soft real-time

The measurement of events and the underlying factors and problems of the control of time in computer systems, introduces another problem and need for synchronization of information, which in most cases, is usually difficult to solve. In version

18 of Erlang an effort was made to rewrite all the aspects responsible for such time control functions, in order to guarantee a monotonous real-time system, despite the system time changing quite frequently. This allowed the software to run with a constant time frequency between launching of events, while acquiring the system time just as it was configured in the operating system.

There were also intimate aspects of language design that needed to be considered in order to avoid another class of problems. Significant aspects such as:

Unique assignments

As in the mathematical statements, the assignment of a value to a variable is done only once and, for the rest of the statement, this variable maintains its immutable value. This guarantees better code tracking and better error detection.

Simple language

The language must have few elements to lower the learning curve. Erlang is a simple language to understand and learn, since it has nothing more than two control structures, excludes loops and employs techniques such as recursiveness and modularisation to achieve small and efficient algorithms. Data structures are also simplified and their power, as in languages such as Prolog or Lisp, are based on lists.

Oriented to the concurrency

As a kind of new way of programming, this language is oriented towards concurrency, hence the most intimate routines of the language itself were designed and prepared to facilitate the realization of concurrent and distributed programs.

Message passing instead of shared memory

One of the main problems of concurrent programming is the execution of critical sections of code for accessing portions of shared memory. This access control ends up being an unavoidable bottleneck. To simplify and try to eliminate as many errors as possible, Erlang/OTP is based on message passing, instead of using known techniques such as traffic lights, or monitors. Message passing makes a process responsible for the data (a critical section in memory) which it is handling and only grants a private view and access of this data, to this process alone. Any other process looking to execute something in that same critical memory section belonging to another process, has to first make a request for the data from the owner process of that memory section. This significantly

abstracts the task of developing concurrent programs, by greatly simplifying the schemes and eliminating the need for explicit blocks.

Not long ago I found a very interesting presentation about Erlang[2], in which it was added, not only everything that Armstrong said his system must have in order to develop the solutions optimally, but also the opposition, why he could not find it in other languages.

At the beginning you have to understand that ***general purpose*** refers to the widespread use of a language to the most common that is usually developed. Obviously, it is more common to make software for the administration of a company than an operating system. The general purpose languages will be optimal for the general development of this business management software and surely not so much for that operating system software. PHP for example, is a fabulous web-oriented language that makes the task much easier for web developers and especially for layout developers who get into the programming field. But it is disastrous for the development of stand-alone servers, because it is designed to be executed and ***die***.

The most widespread languages today, such as C# or Java, present the problem of lacking low-level elements integrated into their systems that allow them to develop concurrent applications in an easy way.

3.History of Erlang

Joe Armstrong attended the Erlang Factory conference in London, in 2010, where he explained the history of Erlang's virtual machine. It's the Erlang/OTP history itself. Using the slides[3] that he provided for the event, we are going to conduct a review of the Erlang/OTP history.

Erlang's idea arose from Ericsson's need to narrow down a problem that had arisen on its AXE platform, which was being developed in PLEX, a proprietary language. Joe Armstrong along with two colleagues, Elshiewy and Robert Virding, developed a concurrent logic of programming for communication channels. This telephony algebra allowed through its notation to describe the Plain Old Telephone Service (POTS) in only fifteen rules.

Through the interest of taking this theory into practice they developed models in Ada, CLU, Smalltalk and Prolog among others. Thus, they discovered that telephone algebra was processed very quickly in high-level systems, that is, in Prolog, therefore they began to develop a deterministic system in it.

[2] http://www.it.uu.se/edu/course/homepage/projektDV/ht05/uppsala.pdf
[3] http://www.erlang-factory.com/upload/presentations/247/erlang_vm_1.pdf

The conclusion reached by the team was that, if a problem can be solved through a series of mathematical equations and by carrying that same scheme to a program so that the functional scheme is respected and understood as it was formulated outside the computational environment, it can be easy to process by people who understand the scheme, even improve it and adapt it. The tests are actually done at the theoretical level on the scheme itself, since algorithmically it is easier to test it with the rules of mathematics than computationally with the number of combinations that it may have.

Prolog was not a language intended for concurrency, so they decided to make one that satisfied all their requirements, based on the advantages they had seen from Prolog to shape their base. Erlang saw the light in 1986, after Joe Armstrong shut himself away to develop the basic idea as an interpreter on Prolog, with a reduced number of instructions that quickly grew thanks to its good reception. Basically, the requirements that were sought to achieve were:

- The processes had to be an intrinsic part of the language, not a library or development framework.
- It should be able to execute from thousands to millions of concurrent processes and each process be independent from the rest, so that if one of them was corrupted it would not damage the memory space of another process. That is, the failure of the processes must be isolated from the rest of the program.
- It must be able to run uninterruptedly, which means that it is not necessary to stop its execution in order to update the system code. Hot swap.

In 1989, the system was beginning to bear fruit, but the problem arose that its performance was not adequate. It was concluded that the language was suitable for the programming that was being carried out, but it would have to be at least 40 times faster.

Mike Williams was responsible for writing the emulator, loader, scheduler and garbage collector (in C language) while Joe Armstrong wrote the compiler, the data structures, the memory heap and the stack; on the other hand, Robert Virding was in charge of writing the libraries. The developed system was optimized to a level where they managed to increase its performance by 120 times than the interpreter did in Prolog.

In the 90s, after having managed to develop products of the AXE range with this language, it was enhanced by adding elements such as distribution, OTP structure, HiPE, bit syntax or compiling pattern matching. Erlang was starting to be a great piece of software, but it had several problems so that it could be widely adopted by the community of programmers. Unfortunately for the development of Erlang, that period

was also the decade of Java and Ericsson decided to focus on ***globally used languages***, so Ericsson forbade further development of Erlang.

Note

HiPE is the acronym for ***High Performance Erlang*** which is the name of a research group on Erlang formed at the University of Uppsala in 1998. The group developed a native code compiler so that Erlang's virtual machine (BEAM) does not have to interpret certain parts of the code if they are already in machine language thus improving their performance.

The imposition of not writing code in Erlang was forgotten over time and the community of Erlang programmers began to grow outside of Ericsson. The OTP team kept developing and supporting Erlang which, in turn, continued as a source of assistance for the HiPE project and applications such as EDoc or Dialyzer.

Prior to 2010, Erlang added capacity for SMP and more recently for multi-core. The 2010 revision of the BEAM emulator runs with a performance 300 times higher than that of the C emulator version, making it 36,000 times faster than the original interpreted in Prolog. More and more sectors are echoing the capabilities of Erlang and more and more companies have started developments on this platform so it is predicted that the use of this language will continue to rise.

4.Developments with Erlang/OTP

The developments in Erlang are increasingly visible to everyone especially in the environment in which Erlang moves: the concurrence and the massive management of events or elements without collapsing or falling. This is an essential and decisive point for companies that have their niche business on the internet and that have gone from selling products to providing services through the network.

In this section we will see the influence of Erlang and how it is settling in the business environment and free software communities as well as the type of implementations that are carried out in both areas.

4.1.Business Sector

Due to the intrinsic advantages of the language and its environment, the creation of real MVC models for web development has become evident.

Necessary items such as ChicagoBoss or Nitrogen deserve mention, whose use can be seen in companies such as the Spanish **Tractis**[4].

[4]Unfortunately this company closed last April 2018.

It is also known the case of **Facebook** that Erlang uses in its chat implementation to support the messages of its 70 million users. Like **Tuenti**, which also used this technology.

The English company **Demonware**[5], specialized in the development and maintenance of infrastructure and server applications for online video games, began using Erlang to support the high number of players in famous video games such as **Call of Duty**.

Several companies in the entertainment sector that make mobile applications have also joined the development of their server applications in Erlang/OTP. An example of this type of companies is **Wooga**[6].

WhatsApp, the currently most relevant application for sending messages between ***smartphones***, uses systems developed in Erlang at the server level and starred in the period of greatest interest on Erlang after its purchase by Facebook in 2014.

One of the archetype companies of Erlang has been Kreditor, which changed its name to **Klarna AB**[7]. This company is dedicated to online payments. This company has the largest staff of programmers in Erlang in the world.

Since the emergence of the ***Cloud*** model, more and more software companies are providing online services instead of selling products, which is why they face widespread use by their users, and even denial of service attacks. These scenarios together with quite heavy services and not very powerful infrastructures make tools like Erlang more and more necessary.

On the website of Erlang Companies[8] you can see a curated list of companies that use Erlang.

4.2.EEF: Erlang Ecosystem Foundation

The Erlang Ecosystem Foundation is a new non-profit organization dedicated to furthering the state of the art for Erlang, Elixir, LFE, and other technologies based on the BEAM. Their goal is to increase the adoption of this sophisticated platform among forward-thinking organizations.

With member-supported Working Groups actively contributing to libraries, tools, and documentation used regularly by individuals and companies relying on the stability and versatility of the ecosystem. The EEF actively invest in critical pieces of technical infrastructure to support

[5] http://www.erlang-factory.com/conference/London2011/speakers/MalcolmDowse
[6] http://es.slideshare.net/wooga/erlang-the-big-switch-in-social-games
[7] https://klarna.com/
[8] https://erlang-companies.org/

its users in their efforts to build the next generation of advanced, reliable, realtime applications.

You can read more about the EFF in their website[9].

4.3.Free Software

There are many samples of great magnitude projects of very diverse nature created on the basis of Erlang. Most of them focus on environments in which great advantage is taken of the management of concurrency and distribution that the Erlang system performs.

Note

Taking advantage of the fact that this list of free software developed in Erlang has been started, the page corresponding to Erlang in Wikipedia has been structured and expanded, so that at this time it will be more extensive than the present list in these pages.

The following list is shown as an example:

Distributed Databases

Apache CouchDB[10]

Is a document based database accessed through HTTP and using the REST format. It is one of the projects that are hosted by the Apache Foundation.

Riak[11]

A NoSQL database inspired by Dynamo (Amazon's NoSQL database). It is used by companies such as Mozilla and Comcast. It is based on an easy scale distribution and completely fault tolerant.

SimpleDB[12]

As indicated by its own website is a non-relational data store with flexible high availability that downloads the work of administration of the databases. That is, a NoSQL system that allows hot swapping of the data schema in an easy way, performs self-indexing and allows the distribution of the data. It was developed by Amazon.

[9] https://erlef.org/
[10] http://couchdb.apache.org
[11] http://www.riak.info/
[12] https://aws.amazon.com/simpledb/

Couchbase[13]

Is a NoSQL database for mission critical systems. With replication, monitoring, fault tolerant and compatible with Memcached.

Web Servers

Yaws[14]

As a complete web server, with the possibility of installing and configuring for it, it only exists Yaws or at least it is the best known in the community. Its configuration is done in a similar way to Apache. It has some quite powerful scripts that run at the server level and allows the use of CGI and FastCGI.

Web Frameworks

ErlyWeb[15]

Has not been modified by Yariv for a few years so its use has declined. Yariv himself used it to make a twitter clone and was initially used for the chat interface for Facebook. Last change in the code was in 2008.

BeepBeep[16]

Is a framework inspired by Rails and Merb although without database integration. Last change in the code was in 2008 and the author archived the repository.

Nitrogen[17]

Is a framework designed to facilitate the construction of web interfaces. It allows us to add HTML code in a simple way and link it with JavaScript with functionality without the need to write a single line of JavaScript code. Last release was in 2015.

N2O[18]

Is a modification of Nitrogen designed to write code on the web asynchronously with the use of websockets. It is becoming famous for its performance and fluidity in the information load.

[13] http://www.couchbase.com/
[14] http://yaws.hyber.org/
[15] https://github.com/yariv/erlyweb
[16] https://github.com/davebryson/beepbeep/
[17] http://nitrogenproject.com/
[18] https://synrc.com/apps/n2o/

ChicagoBoss[19]

Perhaps the most active and complete web framework for Erlang in comparison with the others. It has implementation of views, templates (ErlyDTL), definition of routes, controllers and models through an ORM[20] system.

CMS (Content Management System)

Zotonic[21]

CMS system that allows the design of web pages in a simple way through the programming of the views (DTL) and the management of multimedia content, text and other aspects through the administration interface.

Chat

ejabberd[22]

XMPP server widely used in the Jabber world. This server allows the scaling and management of multi-domains. It is used on sites such as BBC Radio LiveText, Nokia Ovi, KDE Talk, Facebook Chat, Tuenti Chat, LiveJournal Talk, etc.

MongooseIM[23]

Is an ejabberd fork that has been gaining great popularity as it is powered by Erlang Solutions. Its main rewrite was to double its capacity by using binary lists instead of character lists. In recent years the system has been improving a lot in many aspects.

Message Queues

RabbitMQ[24]

A message queuing server widely used in web environment systems with a need for this type of systems for websocket, AJAX or similar connections in which an asynchronous behavior is required over synchronous connections. It was acquired by SpringSource, a subsidiary of VMWare in April 2010.

[19] http://www.chicagoboss.org/
[20] Object Relational Mapping, system used to perform the transformation between objects and tables in order to use directly the objects in code so that the information they handle is stored in a table in the database.
[21] http://zotonic.com/
[22] http://www.ejabberd.im/
[23] https://www.erlang-solutions.com/products/mongooseim.html
[24] http://www.rabbitmq.com/

VerneMQ[25]

Is a message queue server like RabbitMQ but mainly focused on MQTT. According to its creators, it is a bet to obtain a reliable system in the Internet of Things (IoT), which can serve as a monitoring system for storing statistics, mobile messaging, chat system for groups, etc.

5.Erlang and the Concurrency

One of the best proofs that Erlang/OTP works is to show the comparisons that companies like Demonware or people like Joe Armstrong have made. Systems submitted to a test bench to check how they perform in real production or how they could perform in controlled testing environments.

I am going to start by talking about the case of the company Demonware, which I mentioned before in the section about Erlang's use in the business sector, but this time I will detail it with data provided by the company through Malcolm Dowse in the Erlang Factory of London of 2011.

I will continue with a more recent case of the company Riot Games with the production of its servers to support the game League of Legends[26].

Later we will see the test bench that Joe Armstrong made about a service using a couple of Apache and Yaws configurations.

5.1.The case of Demonware

At the conference of Erlang Factory in London, in 2011, Malcolm Dowse, of the company Demonware (from Dublin), gave a presentation entitled Erlang and First-Person Shooters. Tens of millions of Call of Duty Black Ops fans tested Erlang's load.

Demonware is the company that works with Activision and Blizzard giving support to the XBox and PlayStation multi-player game servers. The company was established in 2003 and from that time until 2007 they kept modifying their technology to optimize their servers, until they reached Erlang.

In 2005 they built their infrastructure in C++ and MySQL. Its concurrence of users did not exceed 80 players, fortunately they didn't have to surpass that figure. In addition, the code crashed frequently, which was a serious problem.

[25] https://verne.mq/
[26] http://highscalability.com/blog/2014/10/13/how-league-of-legends-scaled-chat-to-70-million-players-it-t.html

In 2006 the entire business logic was rewritten in Python. It was still maintained internally C++ so the code had become difficult to maintain.

Finally, in 2007, the code of the C++ servers was rewritten to Erlang. It was about 4 months of development with which they got to fix the system so it no longer crashed. They also got to improve and facilitate the configuration of the system (in the C++ version it was necessary to restart to reconfigure, which meant disconnecting all the players). It also provided better log system and administration tools and it became easier to develop new features in many fewer lines of code. By then they had reached 20 thousand concurrent users.

Call of Duty 4 arrived at the end of 2007, which meant a constant growth of users during 5 continuous months. It went from 20 thousand to 2.5 million users. From 500 to 50 thousand requests per second. The company had to expand its nodes from 50 to 1850 servers across several data centers. In the words of Malcolm: ***it was a crisis for the company, we had to grow, without the change to Erlang the crisis could have been a disaster.***

Demonware is one of the companies that has seen the advantages of Erlang. The way in which it implements concurrent programming and the great scalability. Thanks to these factors, they have been able to keep up with the service of the most used and played online games of recent times.

5.2.The case of League of Legends

The Riot Games company was ready for the launch of its video game League of Legends. Keeping in mind the current requirements of any game in which it is necessary to have a chat system and group chat they decided to follow the steps of WhatsApp using ejabberd.

Like WhatsApp, they had to modify ejabberd to avoid bottlenecks and implement the use of specific developments such as the Riak CRDTs[27].

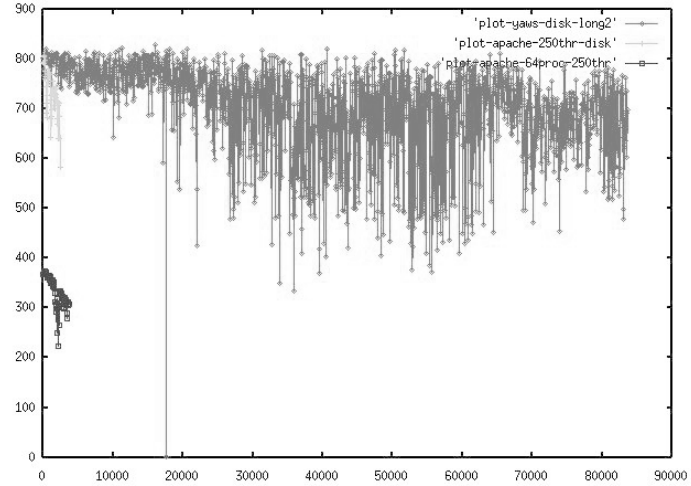
The data shown by the company are quite impressive. Its chat system supports and manages 70 million players. Doing some math based on the numbers that they themselves presented in a highscalability.com article[28], they have 67 million unique players each month, 27 million players every day, 7,5 million concurrent players, a billion routed events per server / day using only 20 -30% of CPU and RAM, 11 thousand messages / second, a few hundred chat servers distributed around the world and managed by only 3 people and finally 99% of uptime.

[27] CRDT stays for Conflict-free Replicated Data Type
[28] http://highscalability.com/blog/2014/10/13/how-league-of-legends-scaled-chat-to-70-million-players-it-t.html

5.3.Yaws vs. Apache

It is already well known the famous graph[29] about the comparison that made Joe Armstrong and Ali Ghodsi between Apache and Yaws. The test is quite easy, on one side, a server, on the other, a client for measurement and 14 clients to generate load.

The proposed test was to generate a Denial of Service (DoS) attack, which made web servers, after receiving an excessive number of requests, degrade their service until they stopped giving it. It is well known that this happens with all systems, since the resources of a server are finite. However, due to its programming, things like the ones shown in the graphic can happen:



In dark gray (marking the point with a circle and occupying the upper lines of the graph) you can see the Yaws response in KB/s scale (Y axis) versus load (X axis). The lines that are cut from the 4 thousand requests correspond to two different configurations of Apache (in black and light gray).

In this case, something similar to that seen with Demonware in the previous section happens, Apache cannot process more than 4000 simultaneous requests, partly due to its integration intimately linked to the operating system, which limits it. However, Yaws remains with the same performance until reaching over 80 thousand simultaneous requests.

[29] http://www.sics.se/~joe/apachevsyaws.html

Erlang is built with its own process management and detached from the operating system. It is usually slower than the one provided by the operating system, but without doubt the scalability and performance that can be achieved help to keep the balance. Each Erlang node can manage a total of about 2 million processes.