Erlang/OTP

VOLUMEN I

Un Mundo Concurrente

3ª edición

MANUEL ÁNGEL RUBIO JIMÉNEZ

Erlang/OTP

Volumen I: Un Mundo Concurrente

Manuel Angel Rubio Jiménez

Erlang/OTP
Volumen I: Un Mundo Concurrente

Manuel Angel Rubio Jiménez

Resumen

El lenguaje de programación Erlang nació en 1986 en los laboratorios Ericsson de la mano de Joe Armstrong. Es un lenguaje funcional con base en Prolog, tolerante a fallos, y orientado al trabajo en tiempo real y a la concurrencia, lo que le proporciona ciertas ventajas en lo que a la declaración de algoritmos se refiere.

Como la mayoría de lenguajes funcionales Erlang requiere un análisis del problema y una forma de diseñar la solución diferente a como se haría en un lenguaje de programación imperativo. Sugiere una mejor y más eficiente forma de llevarlo a cabo. Se basa en una sintaxis más matemática que programática por lo que tiende más a la resolución de problemas que a la ordenación y ejecución de órdenes.

Todo ello hace que Erlang sea un lenguaje muy apropiado para la programación de elementos de misión crítica principalmente a nivel de servidor e incluso para el desarrollo de sistemas embebidos o incrustados por su eficiente gestión de CPU y memoria.

En este libro explicamos el lenguaje y su plataforma, cómo cubre las necesidades para las que fue creado, cómo sacarle el máximo provecho a su forma de realizar las tareas y a su orientación a la concurrencia. Es un repaso desde el principio sobre cómo programar de una forma funcional y concurrente en un entorno distribuido y tolerante a fallos.

Esta tercera revisión comprende hasta la **versión 24** exponiendo la nueva sintaxis para obtener el retorno de pila en una excepción, los alias de procesos, nuevas formas de trabajar con la memoria a través de los atómicos, contadores y términos persistentes, un nuevo capítulo dedicado a ***crypto*** otro capítulo dedicado a ***logger*** y el cambio a **rebar3** para la construcción de nuestros proyectos.

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

Capítulo 1. Lo que debes saber sobre Erlang

Software para un mundo concurrente
— Joe Armstrong

Erlang comienza a ser un entorno y un lenguaje de moda. La existencia creciente de empresas orientadas a la prestación de servicios por Internet con un elevado volumen de transacciones (como videojuegos en red o sistemas de mensajería móvil y chat) hace que en sitios como los Estados Unidos, Reino Unido o Suecia proliferen las ofertas de trabajo que solicitan profesionales en este lenguaje. Existe una necesidad imperiosa de desarrollar entornos con las características de la máquina de Erlang, y la metodología de desarrollo proporcionada por OTP.

En este capítulo introducimos el concepto de Erlang y OTP. Su significado, características e historia. La información de este primer capítulo se completa con las fuentes que lo han motivado y se provee información precisa sobre dónde se ha extraído cada sección.

1. ¿Qué es Erlang?

Para comprender qué es Erlang, debemos entender que se trata de un entorno o plataforma de desarrollo completa. Erlang proporciona no solo el compilador para poder ejecutar el código, sino que posee también una colección de herramientas, y una máquina virtual sobre la que ejecutarlo, por lo tanto existen dos enfoques:

Erlang: el lenguaje

Hay muchas discusiones concernientes a si Erlang es o no un lenguaje funcional. En principio, está entendido que sí lo es, aunque tenga elementos que le hagan salirse de la definición pura. Por ello Erlang podría mejor catalogarse como un lenguaje ***híbrido***, al tener elementos de tipo funcional, de tipo imperativo, e incluso algunos rasgos que permiten cierta orientación a objetos, aunque no completa.

Donde encaja mejor Erlang, al menos desde mi punto de vista, es como un lenguaje orientado a la concurrencia. Erlang tiene una gran facilidad para la programación distribuida, paralela y concurrente y además con mecanismos para la tolerancia a fallos. Fue diseñado desde un inicio para ejecutarse de forma ininterrumpida. Esto significa que se puede cambiar el código de sus aplicaciones sin

detener su ejecución. Más adelante explicaremos cómo funciona esto concretamente.

BEAM: la máquina virtual

Como hemos mencionado antes, Erlang es una plataforma de desarrollo que proporciona no solo un compilador, sino también una máquina virtual para su ejecución. A diferencia de otros lenguajes interpretados como Python, Perl, PHP o Ruby, Erlang se compila y su máquina virtual le proporciona una importante capa de abstracción que le dota de la capacidad de manejar y distribuir procesos entre nodos de forma totalmente transparente (sin el uso de librerías específicas).

La máquina virtual sobre la que se ejecuta el código compilado de Erlang, que le proporciona todas las características de distribución y comunicación de procesos, es también una máquina que interpreta un código máquina<footnote><para>O trozos de código nativo si se emplea HiPE.</footnote> que nada tiene que ver, a ese nivel, con el lenguaje Erlang. Esto ha permitido la proliferación de los lenguajes que emplean BEAM pero no Erlang, como pueden ser: Elixir, LFE, Caramel o Gleam.

Erlang fue propietario hasta 1998 momento en que fue cedido como código abierto (open source) a la comunidad. Fue creado inicialmente por Ericsson, más específicamente por Joe Armstrong, Robert Virding, Bjarne Däcker y Mike Williams inicialmente.

Recibe el nombre de Agnus Kraup Erlang. A veces se piensa que el nombre es una abreviatura de ERicsson LANGuage, debido a su uso intensivo en Ericsson. Según Bjarne Däcker, jefe del Computer Science Lab en su día, esta dualidad es intencionada. Incluso, Robert Virding me aseguró en una ocasión que Ericsson ya disponía de ese nombre mucho antes de concebirse el lenguaje.

2. Características de Erlang

Durante el período en el que Joe Armstrong y sus compañeros estuvieron en los laboratorios de Ericsson, vieron que el desarrollo de aplicaciones basadas en PLEX no era óptimo para la programación de aplicaciones dentro de los sistemas hardware de Ericsson. Por esta razón comenzaron a buscar un sistema de desarrollo óptimo basado en las siguiente premisas:

Distribuido

El sistema debía de ser distribuido para poder balancear su carga entre los sistemas hardware. La idea era poder lanzar procesos no solo en la máquina donde se ejecuta el código sino también en otras máquinas.

Tolerante a fallos

Si una parte del sistema tiene fallos y debe detenerse no debe afectar al resto del sistema. En sus sistemas desarrollados con PLEX un fallo en el código determina una interrupción completa del programa con todos sus hilos y procesos. Incluso capturando excepciones, con el uso de memoria compartida un error puede dejar corrupta esta memoria y afectando por extensión al resto del programa.

Escalable

Los sistemas operativos convencionales tenían problemas en mantener un elevado número de procesos en ejecución. Los sistemas de telefonía que desarrolla Ericsson se basan en tener un proceso por cada llamada entrante, que vaya controlando los estados de la misma y pueda provocar eventos hacia un manejador, a su vez con sus propios procesos. Por lo que se buscaba un sistema que pudiese gestionar desde cientos de miles, hasta millones de procesos.

Cambiar el código en caliente

También es importante en el entorno de Ericsson, y en la mayoría de sistemas críticos o sistemas en producción de cualquier índole, que el sistema no se detenga nunca, aunque haya que realizar actualizaciones. Por ello se agregó también como característica poder cambiar el código en caliente, sin necesidad de parar el sistema y sin afectar al código en ejecución.

Tiempo real blando

La medición de eventos y el problema subyacente del control de la hora en los sistemas informáticos crea un problema de sincronización de eventos difícil de casar en muchos casos. En la versión 18 de Erlang se hizo un esfuerzo por reescribir toda esta parte y garantizar un sistema de tiempo real monótono aún cuando la hora del sistema cambie de forma frecuente permitiendo al software ejecutarse con una frecuencia constante entre lanzamiento de eventos y al mismo tiempo obtener la hora del sistema tal y como esté configurada en el sistema operativo.

También había aspectos íntimos del diseño del lenguaje que se quisieron tener en cuenta para evitar otro tipo de problemas. Aspectos tan significativos como:

Asignaciones únicas

Como en los enunciados matemáticos la asignación de un valor a una variable se hace una única vez y, durante el resto del enunciado,

esta variable mantiene su valor inmutable. Esto nos garantiza un mejor seguimiento del código y una mejor detección de errores.

Lenguaje simple

El lenguaje debe de tener pocos elementos y ninguna excepción para rebajar la curva de aprendizaje. Erlang es un lenguaje simple de comprender y aprender, ya que tiene nada más que dos estructuras de control, carece de bucles y emplea técnicas como la recursividad para conseguir algoritmos pequeños y eficientes. Las estructuras de datos se simplifican también bastante y su potencia se basa en las listas.

Orientado a la Concurrencia

Como una especie de nueva forma de programar, este lenguaje se orienta a la concurrencia de manera que las rutinas más íntimas del propio lenguaje están preparadas para facilitar la realización de programas concurrentes y distribuidos.

Paso de mensajes en lugar de memoria compartida

Uno de los problemas de la programación concurrente es la ejecución de secciones críticas de código para acceso a porciones de memoria compartida. Este control de acceso acaba siendo un cuello de botella ineludible. Para simplificar e intentar eliminar el máximo posible de errores, Erlang/OTP se basa en el paso de mensajes en lugar de emplear técnicas como semáforos o monitores. El paso de mensajes hace que un proceso sea el responsable de los datos y la sección crítica se encuentre solo en este proceso, de modo que cualquiera que pida ejecutar algo de esa sección crítica, tenga que solicitárselo al proceso en cuestión. Esto abstrae al máximo la tarea de desarrollar programas concurrentes simplificando enormemente los esquemas.

Hace tiempo encontré una presentación bastante interesante sobre Erlang[1], en la que se agregaba, no solo todo lo que comentaba Armstrong que debía de tener su sistema para poder desarrollar las soluciones de forma óptima, sino también la contraposición, el porqué no lo pudo encontrar en otros lenguajes.

En principio, debemos el concepto ***propósito general*** que se refiere al uso generalizado de un lenguaje para el desarrollo de cualquier solución. Como es obvio, es más frecuente hacer un software para administración de una empresa que un sistema operativo. Los lenguajes de propósito general serán óptimos para el desarrollo general de ese software de gestión empresarial y seguramente no tanto para ese software del

[1] http://www.it.uu.se/edu/course/homepage/projektDV/ht05/uppsala.pdf

sistema operativo. PHP por ejemplo, es un fabuloso lenguaje orientado a la web que facilita bastante la tarea a los desarrolladores web. Pero es desastroso para el desarrollo de servidores únicos[2].

En sí, los lenguajes más difundidos hoy en día, como C# o Java, presentan el problema de carecer de elementos a bajo nivel integrados en sus sistemas que les permitan desarrollar aplicaciones concurrentes de forma fácil.

3. Historia de Erlang

Joe Armstrong asistió a la conferencia de Erlang Factory de Londres, en 2010, donde explicó la historia de la máquina virtual de Erlang. En sí, es la propia historia de Erlang/OTP. Sirviéndome de las diapositivas[3] que proporcionó para el evento, vamos a dar un repaso a la historia de Erlang/OTP.

La idea de Erlang surgió por la necesidad de Ericsson de acotar un problema que había surgido en su plataforma AXE, que estaba siendo desarrollada en PLEX, un lenguaje propietario. Joe Armstrong junto a dos colegas, Elshiewy y Robert Virding, desarrollaron una lógica concurrente de programación para canales de comunicación. Este álgebra de telefonía permitía a través de su notación describir el sistema público de telefonía (POTS) en tan solo quince reglas.

A través del interés de llevar esta teoría a la práctica desarrollaron modelos en Ada, CLU, Smalltalk y Prolog entre otros. Así descubrieron que el álgebra telefónica se procesaba de forma muy rápida en sistemas de alto nivel, es decir, en Prolog, con lo que comenzaron a desarrollar un sistema determinista en él.

La conclusión a la que llegó el equipo fue que, si se puede resolver un problema a través de una serie de ecuaciones matemáticas y portar ese mismo esquema a un programa de forma que el esquema funcional se respete y entienda tal y como se formuló fuera del entorno computacional, puede ser fácil de tratar por la gente que entiende el esquema, incluso mejorarlo y adaptarlo. Las pruebas realmente se realizan a nivel teórico sobre el propio esquema, ya que algorítmicamente es más fácil de probarlo con las reglas propias de las matemáticas que computacionalmente con la cantidad de combinaciones que pueda tener.

Prolog no era un lenguaje pensado para concurrencia, por lo que se decidieron a realizar uno que satisficiera todos sus requisitos, basándose en las ventajas que habían visto de Prolog para conformar su base. Erlang vio la luz en 1986, después de que Joe Armstrong se encerrase

[2] Conocidos como ***stand-alone***.
[3] http://www.erlang-factory.com/upload/presentations/247/erlang_vm_1.pdf

a desarrollar la idea base como intérprete sobre Prolog, con un número reducido de instrucciones que rápidamente fue creciendo gracias a su buena acogida. Básicamente, los requisitos que se buscaban cumplir eran:

- Los procesos debían ser una parte intrínseca del lenguaje, no una librería o framework de desarrollo.
- Debía poder ejecutar desde miles a millones de procesos concurrentes y cada proceso ser independiente del resto, de modo que si alguno de ellos se corrompiese no dañase el espacio de memoria de otro proceso. Es decir, el fallo de los procesos debe ser aislado del resto del programa.
- Debe poder ejecutarse de modo ininterrumpido, lo que obliga a no detener su ejecución para actualizar el código del sistema. Recarga en caliente.

En 1989, el sistema estaba comenzando a dar sus frutos, pero surgió el problema de que su rendimiento no era el adecuado. Se llegó a la conclusión de que el lenguaje era adecuado para la programación que se realizaba pero tendría que ser unas 40 veces más rápido como mínimo.

Mike Williams se encargó de escribir el emulador, cargador, planificador y recolector de basura (en lenguaje C) mientras que Joe Armstrong escribía el compilador, las estructuras de datos, el heap de memoria y la pila; por su parte Robert Virding se encargaba de escribir las librerías. El sistema desarrollado se optimizó a un nivel en el que consiguieron aumentar su rendimiento en 120 veces de lo que lo hacía el intérprete en Prolog.

En los años 90, tras haber conseguido desarrollar productos de la gama AXE con este lenguaje, se le potenció agregando elementos como distribución, estructura OTP, HiPE, sintaxis de bit o compilación de patrones para ***matching***. Erlang comenzaba a ser una gran pieza de software, pero tenía varios problemas para que pudiera ser adoptado de forma amplia por la comunidad de programadores. Desafortunadamente para el desarrollo de Erlang, aquel periodo fue también la década de Java y Ericsson decidió centrarse en ***lenguajes usados globalmente*** por lo que prohibió seguir desarrollando en Erlang.

Curiosidad

HiPE es el acrónimo de ***High Performance Erlang*** (Erlang de Alto Rendimiento) que es el nombre de un grupo de investigación sobre Erlang formado en la Universidad de Uppsala en 1998. El grupo desarrolló un compilador de código nativo de modo que la máquina virtual de Erlang (BEAM) no tenga que interpretar ciertas partes del código si ya están en lenguaje máquina mejorando así su rendimiento.

Con el tiempo, la imposición de no escribir código en Erlang se fue olvidando y la comunidad de programadores de Erlang comenzó a crecer fuera de Ericsson. El equipo OTP se mantuvo desarrollando y soportando Erlang que, a su vez, continuó sufragando el proyecto HiPE y aplicaciones como EDoc o Dialyzer.

Antes de 2010 Erlang agregó capacidad para SMP y en la década de 2010 más para multi-core. La revisión de 2010 del emulador de BEAM se ejecuta con un rendimiento 300 veces superior al de la versión del emulador en C, por lo que es 36.000 veces más rápido que el original interpretado en Prolog.

Además, en la conferencia de Code BEAM en Estocolmo de 2020, Lukas Larsson presentó una nueva mejora en rendimiento para la versión 24 de Erlang[4]. Un nuevo compilador JIT que potencia la ejecución mostrada con librerías escritas en lenguaje C y en Erlang para tareas con la transformación a JSON y se muestra un rendimiento igual al presentado por el lenguaje C en muchos aspectos.

Cada vez más sectores se hacen eco de las capacidades de Erlang y cada vez más empresas han comenzado desarrollos en esta plataforma por lo que se augura que el uso de este lenguaje siga al alza.

4. Desarrollos con Erlang

Los desarrollos en Erlang cada vez son más visibles para todos sobre todo en el entorno en el que Erlang se mueve: la concurrencia y la gestión masiva de eventos o elementos sin saturarse ni caer. Esto es un punto esencial y decisivo para empresas que tienen su nicho de negocio en Internet y que han pasado de vender productos a proveer servicios a través de la red.

En esta sección veremos la influencia de Erlang y cómo se va asentando en el entorno empresarial y en las comunidades de software libre y el tipo de implementaciones que se realizan en uno y otro ámbito.

4.1. Sector empresarial

La empresa utensils[5] ha puesto a disposición de la comunidad una página donde se recoge un listado actualizado de las empresas, por sectores donde se emplea Erlang hoy en día. Si quieres tener una referencia sobre las empresas puedes acudir a este listado[6] en busca de más información. Por mi parte voy a señalar algunas empresas donde Erlang ha supuesto un cambio importante y se han convertido en verdaderos abanderados del lenguaje y la plataforma.

[4] https://youtu.be/lM7UV95Rpyo
[5] https://utensils.io/
[6] https://erlang-companies.org/

Por ejemplo, la empresa inglesa **Demonware**[7], especializada en el desarrollo y mantenimiento de infraestructura y aplicaciones servidoras para videojuegos en Internet, comenzó a emplear Erlang para poder soportar el número de jugadores de títulos tan afamados como **Call of Duty**.

Varias empresas del sector del entretenimiento que fabrican aplicaciones móviles también se han sumado a desarrollar sus aplicaciones de parte servidora en Erlang/OTP. Un ejemplo de este tipo de empresas es **Wooga**[8].

WhatsApp, la aplicación actualmente más relevante para el intercambio y envío de mensajes entre ***smartphones*** emplea sistemas desarrollados en Erlang a nivel de servidor y protagonizó el período de mayor interés sobre Erlang tras su compra por Facebook en 2014[9].

Una de las empresas estandarte de Erlang ha sido Kreditor, que cambió su nombre a **Klarna AB**[10]. Esta empresa se dedica al pago por Internet y en 7 años ha crecido hasta tener 600 empleados. La mayor plantilla de programadores en Erlang del mundo.

Desde que surgió el modelo ***Cloud*** o _SaaS_[11], cada vez más empresas de software están prestando servicios a través de Internet en lugar de vender productos, por lo que se enfrentan a un uso masificado por parte de sus usuarios, e incluso a ataques de denegación de servicio. Estos escenarios junto con servicios bastante pesados e infraestructuras no muy potentes hacen cada vez más necesarias herramientas como Erlang.

4.2. EEF: Erlang Ecosystem Foundation

La Erlang Ecosystem Foundation[12] es una fundación enfocada a potenciar Erlang y BEAM específicamente. La conforman la mayoría de principales empresas de Erlang y Elixir del mercado como son Ericsson, Erlang Solutions, WhatsApp, Klarna, Cisco, Dashbit o Dockyard. Tienen muchos grupos de trabajo dedicados a potenciar la seguridad, el marketing, los sistemas embebidos o incluso recientemente Machine Learning.

4.3. Software libre

Hay muchas muestras de proyectos de gran envergadura de muy diversa índole creados en base a Erlang. La mayoría de ellos se centra en

[7] http://www.erlang-factory.com/conference/London2011/speakers/MalcolmDowse
[8] http://es.slideshare.net/wooga/erlang-the-big-switch-in-social-games
[9] http://altenwald.org/2014/02/22/erlang-y-whatsapp/
[10] https://klarna.com/
[11] Software as a Service, o Software como Servicio.
[12] https://erlef.org/

entornos en los que se saca gran ventaja de la gestión de concurrencia y distribución que realiza el sistema de Erlang.

Nota

Aprovechando que se ha comenzado a hacer esta lista de software libre desarrollado en Erlang se ha estructurado y ampliado la página correspondiente a Erlang en Wikipedia (en inglés de momento y poco a poco en castellano), por lo que en estos momentos será más extensa que la lista presente en estas páginas.

El siguiente listado se muestra como ejemplo:

- **Base de Datos Distribuidas**

  - Apache CouchDB[13], es una base de datos documental con acceso a datos mediante HTTP y empleando el formato REST. Es uno de los proyectos que están acogidos en la fundación Apache.

  - Riak[14], una base de datos NoSQL inspirada en Dynamo (la base de datos NoSQL de Amazon). Es usada por empresas como Mozilla y Comcast. Se basa en una distribución de fácil escalado y completamente tolerante a fallos.

  - SimpleDB[15], tal y como indica su propia web (en castellano) es un almacén de datos no relacionales de alta disponibilidad flexible que descarga el trabajo de administración de las bases de datos. Es decir, un sistema NoSQL que permite el cambio en caliente del esquema de datos de forma fácil que realiza auto-indexación y permite la distribución de los datos. Fue desarrollada por Amazon.

  - Couchbase[16], es una base de datos NoSQL para sistemas de misión crítica. Con replicación, monitorización, tolerante a fallos y compatible con Memcached.

  - DalmatinerDB[17], es una base de datos de series de tiempo pensada para el almacenamiento de estadísticas o información para ser representada en gráficos y tenga un carácter temporal.

- **Servidores Web**

[13] http://couchdb.apache.org
[14] http://wiki.basho.com/Riak.html
[15] http://aws.amazon.com/es/simpledb/
[16] http://www.couchbase.com/
[17] https://dalmatiner.io/

  - Yaws[18]. Como servidor web completo, con posibilidad de instalarse y configurarse para ello, solo existe (al menos es el más conocido en la comunidad) Yaws. Su configuración se realiza de forma bastante similar a Apache. Tiene unos scripts que se ejecutan a nivel de servidor bastante potentes y permite el uso de CGI y FastCGI.

- **Frameworks Web**

  - Nitrogen[19], es un framework pensado para facilitar la construcción de interfaces web. Nos permite agregar código HTML de una forma simple y enlazarlo con funcionalidad de JavaScript sin necesidad de escribir ni una sola línea de código JavaScript. Aunque lleva algo parado desde 2015.

  - N2O[20], es una modificación de Nitrogen pensada para escribir código en la web de forma asíncrona con el empleo de websockets. Comienza a ser famoso por su rendimiento y fluidez en la carga de información.

  - ChicagoBoss[21], quizás el más activo y completo de los frameworks web para Erlang a día de hoy. Tiene implementación de vistas, plantillas (ErlyDTL), definición de rutas, controladores y modelos a través de un sistema ORM[22].

  - NovaFramework[23], es un framework bastante reciente que se basa únicamente en proporcionar y facilitar la parte del controlador y rutas de un proyecto web. Con facilidades para WebSocket.

- **CMS** (Content Management System)

  - Zotonic[24], sistema CMS[25] que permite el diseño de páginas web de forma sencilla a través de la programación de las vistas (DTL) y la gestión del contenido multimedia, texto y otros aspectos a través del interfaz de administración.

- **Chat**

[18] http://yaws.hyber.org/
[19] http://nitrogenproject.com/
[20] https://n2o.space/
[21] http://www.chicagoboss.org/
[22] ***Object Relational Mapping***, sistema empleado para realizar la transformación entre objetos y tablas para emplear directamente los objetos en código y que la información que estos manejen se almacene en una tabla de la base de datos.
[23] http://novaframework.org/
[24] http://zotonic.com/
[25] ***Content Management System***, Sistema de Administración de Contenido

- ejabberd[26], servidor de XMPP muy utilizado en el mundo Jabber. Este servidor permite el escalado y la gestión de multi-dominios. Es usado en sitios como la BBC Radio LiveText, Ovi de Nokia, KDE Talk, Chat de Facebook, Chat de Tuenti, LiveJournal Talk, etc.

- MongooseIM[27] es un fork de ejabberd que ha ido obteniendo gran popularidad por ser impulsado por Erlang Solutions. Su principal reescritura fue para duplicar su capacidad al emplear listas binarias en lugar de listas de caracteres. En estos últimos años han ido mejorando mucho el sistema en muchos aspectos. No obstante, también han eliminado algunas características que aún no han conseguido reemplazar como PubSub.

- **Colas de Mensajes**

  - RabbitMQ[28], servidor de cola de mensajes muy utilizado en sistemas de entornos web con necesidad de este tipo de sistemas para conexiones de tipo WebSocket, AJAX o similar en la que se haga necesario un comportamiento asíncrono sobre las conexiones síncronas. Fue adquirido por SpringSource, una filial de VMWare en abril de 2010.

  - VerneMQ[29], es un servidor de cola de mensajes como RabbitMQ pero enfocado principalmente en MQTT. Según sus creadores es una apuesta para obtener un sistema fiable en la Internet de las Cosas (IoT) pudiendo servir como sistema para monitorización almacenando estadísticas, mensajería móvil, sistema de chat para grupos, etc.

5. Erlang y la Concurrencia

Una de las mejores pruebas de que Erlang/OTP funciona, es mostrar las comparaciones que empresas como Demonware o gente como el propio Joe Armstrong han realizado. Sistemas sometidos a un banco de pruebas para comprobar cómo rinden en producción real o cómo podrían rendir en entornos de pruebas controlados.

Comenzaré por comentar el caso de la empresa Demonware, de la que ya comenté algo en la sección de uso de Erlang en el ***Sector empresarial***, pero esta vez lo detallaré con datos que aportó la propia compañía a través de Malcolm Dowse en la Erlang Factory de Londres de 2011.

[26] http://www.ejabberd.im/
[27] https://www.erlang-solutions.com/products/mongooseim.html
[28] http://www.rabbitmq.com/
[29] https://verne.mq/

Seguiré con un caso más reciente de la empresa Riot Games con la puesta en producción de sus servidores para soportar el juego League of Legends[30].

Después veremos el banco de pruebas que realizó Joe Armstrong sobre un servicio empleando un par de configuraciones de Apache y Yaws, y una nueva versión realizada por mi en julio de 2017 agregando además a Nginx en la comparación.

5.1. El caso de Demonware

En la conferencia de Erlang Factory de Londres, en 2011, Malcolm Dowse, de la empresa Demoware (de Dublín), dictó una ponencia titulada ***Erlang and First-Person Shooters*** (Erlang y los Juegos en Primera Persona). Decenas de millones de fans de Call of Duty Black Ops probaron la carga de Erlang.

Demonware es la empresa que trabaja con Activision y Blizzard dando soporte de los servidores de juegos multi-jugador XBox y PlayStation. La empresa se constituyó en 2003 y desde esa época hasta 2007 se mantuvieron modificando su tecnología para optimizar sus servidores, hasta llegar a Erlang.

En 2005 construyeron su infraestructura en C++ y MySQL. Su concurrencia de usuarios no superaba los 80 jugadores, afortunadamente no se vieron en la situación de superar esa cifra. Además, el código se colgaba con frecuencia, lo que suponía un grave problema.

En 2006 se reescribió toda la lógica de negocio en Python. Se seguía manteniendo a nivel interno C++ con lo que el código se había hecho difícil de mantener.

Finalmente, en 2007, se reescribió el código de los servidores de C con Erlang. Fueron unos 4 meses de desarrollo con el que consiguieron que el sistema ya no se colgase, que se mejorase y facilitase la configuración del sistema (en la versión C era necesario reiniciar para reconfigurar, lo que implicaba desconectar a todos los jugadores). También se dotó de mejores herramientas de log y administración y se hacía más fácil desarrollar nuevas características en muchas menos líneas de código. Para entonces habían llegado a los 20 mil usuarios concurrentes.

A finales de 2007 llegó ***Call of Duty 4***, que supuso un crecimiento constante de usuarios durante 5 meses continuados. Se pasó de 20 mil a 2,5 millones de usuarios. De 500 a 50 mil peticiones por segundo. La empresa tuvo que ampliar su nodo de 50 a 1850 servidores en varios

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

centros de datos. En palabras de Malcolm: ***fue una crisis para la compañía, teníamos que crecer, sin el cambio a Erlang la crisis podría haber sido un desastre***.

Demonware es una de las empresas que ha visto las ventajas de Erlang. La forma en la que implementa la programación concurrente y la gran capacidad de escalabilidad. Gracias a estos factores, han podido estar a la altura de prestar el servicio de los juegos en línea más usados y jugados de los últimos tiempos.

5.2. El caso de League of Legends

La empresa Riot Games estaba preparada para el lanzamiento de su videojuego League of Legends. Teniendo en mente los requisitos actuales de cualquier juego en el que se hace necesario tener un sistema de chat y chat de grupo decidieron seguir los pasos de Whatsapp empleando ejabberd.

Al igual que Whatsapp, tuvieron que modificar ejabberd para evitar los cuellos de botella e implementar el uso de de desarrollos específicos como el de Riak CRDTs (Tipos de Datos Replicados y Convergentes).

Los datos mostrados por la empresa son bastante impresionantes. Su sistema de chat soporta y maneja 70 millones de jugadores. Haciendo algunos números que ellos mismos presentaron en un artículo de highscalability.com[31], tienen cada mes 67 millones de jugadores únicos, 27 millones de jugadores cada día, 7.5 millones de jugadores concurrentes, mil millones de eventos enrutados por servidor/día usando solo el 20-30% de CPU y RAM, 11 mil mensajes/segundo, unos cuantos cientos de servidores de chat distribuidos por el mundo y manejados por solamente 3 personas y finalmente 99% de tiempo de disponibilidad (***uptime***).

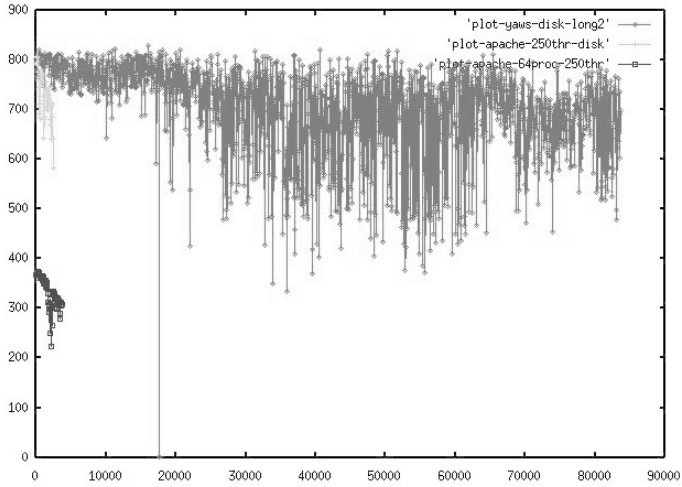
5.3. Yaws contra Apache y Nginx

Es bastante conocido ya el famoso gráfico[32] sobre la comparativa que realizaron Joe Armstrong y Ali Ghodsi entre Apache y Yaws. La prueba es bastante fácil, de un lado, un servidor, de otro, un cliente para medición y 14 clientes para generar carga.

La prueba propuesta era generar un ataque de denegación de servicio (DoS), que hiciera que los servidores web, al recibir un número de peticiones excesivo, fuesen degradando su servicio hasta dejar de darlo. Es bien conocido que este hecho pasa con todos los sistemas, ya que los

[31] http://highscalability.com/blog/2014/10/13/how-league-of-legends-scaled-chat-to-70-million-players-it-t.html
[32] http://www.sics.se/~joe/apachevsyaws.html

recursos de un servidor son finitos. No obstante, por su programación, pueden pasar cosas como las que se visualizan en el gráfico:



En gris oscuro (marcando el punto con un círculo y ocupando las líneas superiores del gráfico) puede verse la respuesta de Yaws en escala de KB/s (eje Y) frente a carga (eje X). Las líneas que se cortan a partir de las 4 mil peticiones corresponden a dos configuraciones diferentes de Apache (en negro y gris claro).

En este caso, pasa algo parecido a lo visto con Demonware en la sección anterior, Apache no puede procesar más de 4000 peticiones simultáneas, en parte debido a su integración íntimamente ligada al sistema operativo, que le limita. Sin embargo, Yaws se mantiene con el mismo rendimiento hasta llegar a superar las 80 mil peticiones simultáneas.

Erlang está construido con gestión de procesos propia y desligada del sistema operativo. En sí, suele ser más lenta que la que proporciona el sistema operativo, pero sin duda la escalabilidad y el rendimiento que se consigue pueden paliar ese hecho. Cada nodo de Erlang puede manejar en total unos 2 millones de procesos.

En julio de 2017 hice una prueba similar empleando Nginx, Apache y Yaws para las pruebas. Los datos que extraje los puedes revisar en este artículo en el blog de Altenwald:

https://altenwald.org/2017/07/09/yaws-probando-servidor-web-erlang/

Cuando realicé las pruebas que quedé satisfecho. Demostraban un hecho defendido desde hace años con Erlang, la garantía de obtener respuestas correctas tal y como se espera de sistemas reactivos.

Las pruebas nos dejan con la conclusión de si vas a emplear páginas estáticas o generadas con sistemas como Jekyll, Octopress o Lambdapad[33] y quieres tener tu sistema reactivo dando respuestas correctas Yaws es una opción ineludible.

[33] https://lambdapad.com

Erlang/OTP

Volumen I: Un Mundo Concurrente

Manuel Angel Rubio Jiménez