|  |
| --- |
|  |
| Proyecto: Sistemas Distribuidos - RPC |
| goChar – Reconocimiento de caracteres distribuido |

|  |
| --- |
| David Barranco Alfonso  http://jimplush.com/talk/wp-content/uploads/2015/11/68747470733a2f2f7261772e6769746875622e636f6d2f676f6c616e672d73616d706c65732f676f706865722d766563746f722f6d61737465722f676f706865722e706e67.pngJesús Bocanegra Linares  26-4-2016 |

Contenido

[Introducción 2](#_Toc449563967)

[Lenguaje Go. De dónde viene y hacia dónde va 3](#_Toc449563968)

[Paquetes de GO 4](#_Toc449563969)

[Reconocimiento de caracteres en Python 5](#_Toc449563970)

# Introducción

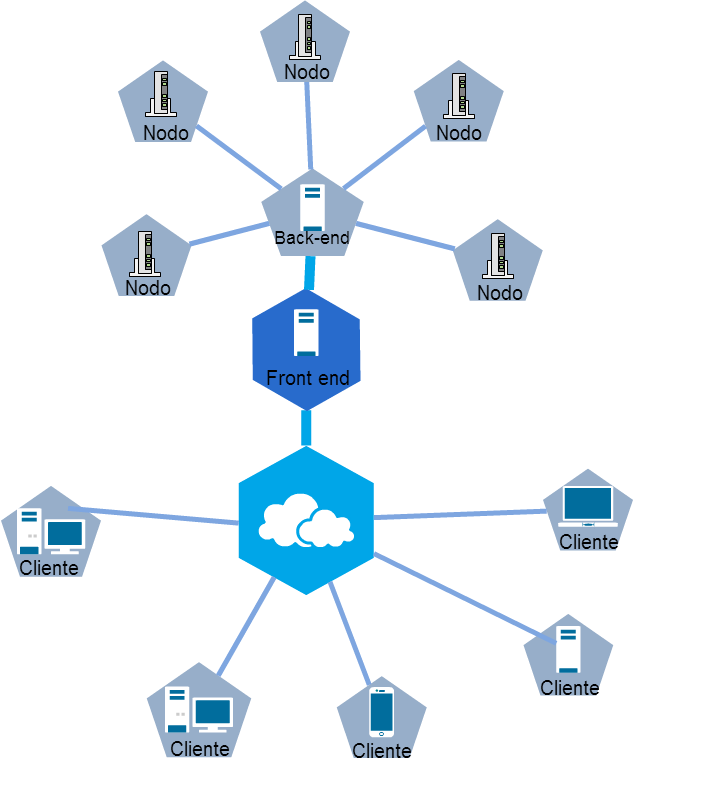
Este es un proyecto que se ha realizado para la asignatura de **sistemas distribuidos de 4º de GITT.**  En esta asignatura se propone realizar un proyecto basado en las **Remote Procedure Calling (RPC), o llamadas a procedimientos remotos.**

Una llamada a procedimiento remoto consiste en un protocolo que permite a un software o programa ejecutar código en otra máquina remota sin tener que preocuparse por la comunicación. Las RPC son muy utilizadas dentro de la comunicación cliente-servidor, siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando este de vuelta el resultado de dicha operación al cliente.

En nuestro caso hemos decidido realizar un sistema de reconocimiento de caracteres óptico distribuido. Un **sistema de reconocimiento de caracteres óptico (OCR en inglés)** es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos. El proceso básico que se lleva a cabo es convertir el texto que aparece en una imagen en un archivo de texto que podrá ser editado y utilizado como tal por cualquier otro programa o aplicación que lo necesite. Partiendo de una imagen, el reconocimiento de estos caracteres se suele realizar comparándolos con unos patrones o plantillas que contienen todos los posibles caracteres.

Partiendo de la premisa de la asignatura de realizar un proyecto basado en las **Remote Procedure Calling** hemos decidido realizar las operaciones de reconocimiento de caracteres de forma distribuida. Es decir, la estructura básica del proyecto (sobre la que profundizaremos después) es la siguiente:

Muchos clientes desde distintas plataformas pueden acceder al servidor front-end que sirve una página web donde al usuario se le presenta una interfaz donde puede subir una imagen para reconocer un carácter.



La imagen se envía mediante un HTTP POST al servidor back-end, distribuye la carga de trabajo a uno de los nodos que tenga libre para procesar la imagen. Estos nodos ejecutan un script en **Python** programado usando la librería **TensorFlow**. Gracias a esta son capaces de descifrar el carácter que contiene la imagen usando técnicas de redes neuronales convolucionales. Una vez conseguido este carácter, la respuesta se envía de vuelta al servidor back-end. De esta forma el usuario puede ver el resultado de la operación en su navegador.

Ya que es un proyecto donde la heterogeneidad de las plataformas donde se va a trabajar es evidente (un nodo puede ser un ordenador con Windows, una Raspberry Pi con una distribución Linux o un servidor basado en Mac OSX, por ejemplo) hemos decidido utilizar el lenguaje **Go**.

# Lenguaje Go. De dónde viene y hacia dónde va

Como se ha comentado antes, la heterogeneidad de las plataformas usadas en este proyecto es evidente. Debido a que los nodos encargados del procesamiento de las imágenes pueden estar ubicados en distintos tipos de máquinas con distintos tipos de sistemas operativos y distintas arquitecturas de procesador es conveniente basar toda la parte interna del proyecto en un lenguaje que pueda ser compilado para diferentes tipos de máquinas de manera relativamente fácil.

**Go** es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de **C**. Ha sido desarrollado por Google y sus diseñadores iniciales son:

****



**Robert Griesemer Ken Thompson Rob Pike**

-V8 Javascript Engine -Multics OS (60’s) -Plan9

-Diseño de Java Hotspot -Unix -UTF-8

-Strongtalk -Lenguaje B , UTF-8

Se puede decir que “nació” en 2009 y es un lenguaje al que hoy en día están migrando sus servidores muchas empresas (Twitter, Docker, Google, Tumblr…).

Algunas de sus características básicas:

* *Compilado*: No se necesita instalar ningún programa para que el programa que desarrolles funcione en el sistema operativo para el que lo compilaste.
* *Estáticamente Tipado*: Las variables son tipadas de manera estática, así que, si la variable **x** se definie como entera, será entera durante todo su alcance.
* *Concurrente*: Está inspirado en CSP (Communicating sequential processes — de Charles Antony Hoare, ganador del premio Turing en 1980).
* *Uso poco usual de POO*: Go no usa clases, no usa herencia y el uso de interfaces se realiza de manera implícita. Esto con el fin de mejorar el rendimiento al momento de diseñar el software.

Es por lo anterior que su aprendizaje es bastante sencillo, sobre todo viniendo de lenguajes de programación orientada a objetos como C++ o JAVA.

## Paquetes de GO

En Go no se trabaja con librerías, se trabaja con paquetes. Estos están formados por varios archivos “.go” los cuales ayudan a organizar el código y añaden funcionalidades.

Gracias a la gran comunidad de desarrolladores que hay detrás de Go puedes encontrar un sinfín de paquetes dedicados a tareas muy específicas. Por ejemplo, el paquete **resize** se encarga de ofrecer funciones para el redimensionamiento de imágenes, apoyándose en los paquetes de procesamiento de imágenes que ofrece Go.

Dado que existen muchos desarrolladores detrás de Go creando nuevos paquetes para ofrecer nuevas funcionalidades, Go ha de tener una gran integración con **GitHub** para poder integrar paquetes sin tener que descargar el repositorio del desarrollador.

Si quisiéramos añadir a nuestro software las funciones de redimensionamiento de imágenes, deberíamos hacer:

$Go get github.com/nfnt/resize //añadimos el paquete para trabajar con el

//Y en el código…

Import (

“github.com/nfnt/resize”

)

Para la realización del proyecto hemos estudiado varios paquetes con los que implementar el sistema de conexión entre el servidor backend (que a su vez hace de maestro) y los esclavos:

* Paquete net/rpc: Es el paquete básico que trae golang para trabajar con las rpc’s. No implementa comunicación bidireccional entre 2 máquinas, con lo que imposibilitaba la idea que teníamos de que el maestro y esclavo fueran servidor y cliente a la vez. Fue descartado.
* Paquete [goRPC, creado por valyala](https://github.com/valyala/gorpc): Paquete basado en **net/rpc.** Añade muchas funcionalidades no incluidas en el paquete por defecto de rpc y simplifica bastante la creación de servidor y cliente. Entre algunas de estas funcionalidades que hizo decantarnos por este.
  + - Minimiza el número de llamadas al sistema
    - Minimiza la carga de CPU
    - Minimiza el número de conexiones TCP
    - Soporta cargas de > 40.000 peticiones por segundo

La primera versión del proyecto se creó con este paquete. Cuando comenzamos las pruebas, descubrimos que no soporta comunicaciones bidireccionales. Por lo que tuvimos que descartar el proyecto y buscar otro paquete que si implementara esto.

* Paquete [rpc2, creado por cenk](https://github.com/cenk/rpc2): Al igual que **goRPC** está basado en el paquete net/rpc. El principal objetivo de este paquete es ser **simple** e implementar **comunicaciones bidireccionales**. Fue el paquete con el que terminamos realizando el proyecto. Simplifica incluso más la creación de un cliente y un servidor para conexiones RPC, algo que nos venía bastante bien viendo el tiempo que nos quedaba hasta la entrega del proyecto.

Otros paquetes, quizás con menos importancia que los anteriores debido al propósito del proyecto fueron los siguientes:

* **Container/list:** Para trabajar con listas enlazadas
* **Image:** Para trabajar con imágenes
* **Net/http:** Para la creación en el backend de un servidor http
* **Image/color:** Para la conversión de imágenes a escala de grises
* **Math:** Operaciones sobre los píxeles para la conversión de estas a escala de grises
* **Log:** Impresión de logs en la consola del servidor/cliente
* **Net:** Aportaba funciones de red
* **Strconv:** Conversión de cadenas

# Reconocimiento de caracteres en Python

Blabla

//Despues de tensorFlow viene el de la estructura del proyecto.

# Estructura del proyecto

## Servidor Front-end

## Servidor Back-end - Maestro

## Esclavo

# Ejecución del proyecto

# Herramientas utilizadas

Para el desarrollo de este proyecto, las herramientas que hemos utilizado han sido:

* **LiteIde**: Pequeño IDE para programar en Go.
* **Github**: Para control de código
* **Dropbox**: Para compartición de archivos

Meh.

# Conclusiones y trabajo futuro