Interfaz web para la visualización 3D y segmentación interactiva de imágenes científicas utilizando un sistema distribuido

PROYECTO DE SISTEMAS INFORMÁTICOS

Pedro Javier Rodríguez Rodrigo

Dirigido por:

José Luis Vázquez-Poletti & José Manuel Velasco

Facultad de Informática

Universidad Complutense de Madrid

Documento maquetado con TEXIS v.1.0.

Interfaz web para la visualización 3D y segmentación interactiva de imágenes científicas utilizando un sistema distribuido

Departamento de Arquitectura de Computadores y Automática

Dirigido por:

José Luis Vázquez-Poletti & José Manuel Velasco

Facultad de Informática

Universidad Complutense de Madrid

Junio 2015

Autorización

Yo, Pedro Javier Rodríguez Rodrigo, autorizo a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a su autor, tanto este documento como el código, la documentación y/o el prototipo desarrollado.

Firma:

Madrid, a de de

Abstract / Resumen

The presence of technology in our day to day activities, coupled with the reduction in data storage costs, has started the era of Big Data. However, processing and visualizing such big quantities of data require especific software or hardware, and not all the users have the opportunity to access this technologies. Thanks to the newest cloud computing technologies it is possible to provide a service that will allow every user to analyze, operate and visualize his data as fast as possible using the latest technologies. The user will interact with the application using a web browser. The service will get as input the user data, and will provide different ways to analyze it. The user will not need any kind of knowledge about the technologies used to work with the data. This project designs a distributed system, capable of providing the service to several users at the same time.

La presencia de la tecnología en la mayoría de las actividades que realizamos a diario, junto con la disminución del coste de almacenamiento de los datos que se generan, ha dado paso a la era del Big Data. Pero el procesado y visualización de grandes cantidades de datos requiere de hardware o software específico que puede estar fuera del alcance de algunos usuarios. Gracias a las nuevas tecnologías de computación en la nube es posible proporcionar un servicio mediante el cual todos los usuarios puedan visualizar y analizar sus datos sin preocuparse de adquirir hardware o software específico. El usuario dispondrá de una interfaz web para interaccionar con el servicio. Este recibirá los datos del usuario y le proporcionará varias opciones para analizarlos, sin que el usuario necesite tener ningún tipo de conocimiento de las tecnologías utilizadas. Este proyecto diseña un sistema distribuido que permita proveer el servicio a varios usuarios al mismo tiempo.

Keywords / Palabras clave

Keywords

Big Data, cloud computing, data visualization, distributed system, 3D graphics, Cycles render engine.

Palabras Clave

Big Data, computación en la nube, visualización de datos, sistema distribuido, gráficos 3D, motor de renderizado Cycles.

Agradecimientos

Sobre todo lo demás, a mis padres, puesto que han sido ellos los que me han dado la oportunidad de estudiar lo que siempre quise. Y a mi hermana, que junto con ellos siempre me han apoyado en las cosas que he hecho.

A mi novia, que ha tenido que sufrir mis nervios cuando se acercaba la fecha de entrega y, aún así, siempre me ha dado ánimos.

A los dos profesores que me han guiado durante el desarrollo de este proyecto, José Luis y José Manuel, porque sin sus ideas y apoyo hubiera sido difícil sacarlo adelante.

A las personas que han desarrollado las distintas tecnologías que he utilizado, y que han compartido con todo el mundo su esfuerzo y dedicación. Entre ellos, Marco Antonio y Pedro Pablo Gómez Martín, creadores de TEXIS, la plantilla utilizada para redactar esta memoria.

Y por último agradecer a todas las personas que han compartido sus conocimientos a través de diferentes sitios en la Red, y que me han ayudado a avanzar.

Muchas gracias.

Índice

Capítulo 1

Introducción

La tecnología está presente en la mayoría de las actividades que realizamos a lo largo del día. Cada vez que utilizamos los servicios de una empresa, por pequeña que sea, le estamos proporcionando datos que, bien utilizados, pueden servirles para mejorar su negocio. Las empresas grandes lo saben muy bien, y por eso algunas basan parte de su negocio en esto mismo. Por eso, no es difícil encontrar servicios que se ofrecen al público de manera gratuita, independientemente de que esos servicios tienen unos costes muy altos para la empresa que los proporciona.

¿Por qué una empresa ofrecería un servicio de manera gratuita cuando ese servicio le genera unos gastos? En la mayoría de los casos, la empresa que proporciona el servicio utiliza la información que obtiene de ellos para hacer rentable el servicio. Esa información puede convertirse en dinero de varias maneras, entre las cuales están la de vender la misma, o analizarla y utilizar los resultados para mejorar ese u otros negocios de la empresa.

El problema radica en realizar el análisis de esa información. Las grandes empresas pueden disponer del hardware, el software y las personas con los conocimientos adecuados para analizar todos esos datos, pero las pequeñas y medianas empresas no siempre tienen los medios adecuados para hacerlo.

Con este proyecto se pretende diseñar un servicio que permita a sus usuarios generar gráficas y analizar los datos de una manera sencilla, sin que el

usuario final necesite tener conocimientos específicos de cómo hacerlo. Además, el servicio se ofrecerá a través de una interfaz web, liberando al usuario de la necesidad de disponer del hardware o software necesario para el tratamiento de los datos.

El sistema que proporcionará el servicio al usuario funcionará de manera distribuida entre diferentes máquinas, encargadas cada una de ellas de distintas tareas. De esta manera el trabajo de unos usuarios no afectará al sistema completo y, por tanto, no afectará a los demás usuarios del mismo.

Capítulo 2

Estado actual

Resumen: En este capítulo se trata el estado actual de diversas tecnologías relacionadas con el proyecto y de los motivos que han llevado a descartarlas o usarlas en el mismo. También se hablará de algunos conceptos que están relacionados con el proyecto.

2.1. Tecnologías

2.1.1. Python

Python es un lenguaje de programación interpretado que se creó en torno a 1990. Durante la última década su uso se ha disparado, sobre todo dentro del ámbito. Debido a esto, el lenguaje dispone de una gran cantidad de módulos y librerías para realizar todo tipo de tareas. Además, determinadas características del lenguaje como que sea un lenguaje orientado a objetos, o de tipado débil, hacen de él un lenguaje muy versátil a la hora de implementar prototipos de aplicaciones, porque te permite un desarrollo rápido.

Por los motivos anteriormente citados, desde un principio se decidió usar Python como lenguaje para este proyecto. Python y sus librerías facilitan mucho la interacción entre todas las partes de las que se compone este sistema y, dado que el trabajo de renderizado final lo desarrolla un software

completamente ajeno a Python (el motor de renderizado Cycles), el uso de otro lenguaje, compilado o no, podría no significar un aumento notable en la eficacia del sistema global.

2.1.2. Librerías gráficas

Existe una gran cantidad de librerías gráficas que nos permiten analizar datos y generar gráficos para facilitar su comprensión. La mayoría de estas librerías están desarrolladas por organizaciones académicas o investigadores que necesitan una manera de facilitar la visualización de datos a la que se enfrentan en su trabajo diario.

Sin embargo, estas librerías requieren de conocimientos del lenguaje para usarlas y, en la mayoría de los casos, de una máquina potente donde ejecutar los programas que van a tratar los datos y a generar los gráficos. Además, estas librerías suelen estar pensadas para que sus gráficos sean presentados simplemente en la pantalla del ordenador donde se han generado, o para que se incluyan en publicaciones científicas. Por esto último, no es común encontrar librerías de este tipo que permitan generar gráficos en sistemas que no tienen ninguna pantalla (como es común en el caso de los servidores que se utilizan para el tratamiento de datos), o están limitados por la resolución máxima de la pantalla del sistema.

A continuación un par de ejemplos de librerías disponibles en Python:

- Matplotlib (http://matplotlib.org/)
  Es probablemente la más utilizada en Python, y se especializa en la generación de gráficos para su uso en publicaciones científicas. Genera imágenes de los gráficos que se pueden usar en cualquier otro sitio, pero está muy dirigida a gráficos en 2D y las opciones de gráficos tridimensionales son bastante limitadas (aunque con el tiempo van mejorando poco a poco).

- Vispy (http://vispy.org/)
  Esta librería está siendo desarrollada por un grupo de personas que anteriormente trabajaban en varias librerías independientes y que han unificado gran parte del trabajo que habían hecho por separado. Esta se centra en la visualización de datos en tiempo real, aprovechándose de la GPU (*Graphics Processing Unit*) para generar los gráficos más rápido. Aunque también dispone de algunos gráficos en 3D, aún está muy limitada, y no está pensada para su uso en sistemas sin pantalla o en sistemas distribuidos. En unos años, esta librería podría ser una muy buena opción a la hora de visualizar grandes cantidades de datos en tiempo real.

2.1.3. Programas de renderizado

En la actualidad existe en el mercado una gran cantidad de software para el renderizado de imágenes tridimensionales. Casi todos los productos de este tipo suelen tener precios muy elevados, sin embargo, los resultados que producen son imágenes de una calidad muy alta.

Algunos de los programas comerciales de renderizado más conocidos son:

- Luxrender (http://www.luxrender.net)
- Mitsuba (http://www.mitsuba-renderer.org/)
- RenderMan (http://renderman.pixar.com/)

Desde hace relativamente poco, se está desarrollando un nuevo software de renderizado de código abierto, que es el que finalmente se decidió usar en este proyecto. Este software, de nombre “Cycles”, es uno de los motores de renderizado de los que dispone “Blender” (https://www.blender.org/). Las últimas actualizaciones que se han llevado a cabo en su desarrollo lo han convertido en un programa independiente que no necesita de “Blender” para funcionar y que no exige que el sistema en el que funcione tenga conectada una pantalla.

Utilizar este software proporciona varias ventajas:

- Permite instalarlo en servidores dedicados que formen parte de un sistema distribuido.
- El renderizador es completamente independiente de otros programas, y trabaja en base a archivos XML con los datos sobre la escena.
- Es un software en continuo desarrollo, por lo tanto a futuro podría disponer de nuevas características que beneficiasen este proyecto.

2.1.4. Programas de renderizado o librerías gráficas

El uso de programas de modelado, iluminación y renderizado de gráficos tridimensionales en lugar de las librerías de gráficos de los lenguajes de programación, nos permite disponer del control total sobre lo que queremos representar en nuestra escena y la manera en la que queremos que se haga (objetos, luces, cámara, etc.) Estos programas están diseñados para generar imágenes que en muchas ocasiones serán de una resolución mayor a la que permite la tarjeta gráfica del sistema y facilitan el uso de un sistema distribuido a la hora de crear una única imagen. Algunos permiten su uso sin ni siquiera tener una pantalla conectada al sistema, lo que los hace ideales para su uso en servidores que formen parte de un sistema distribuido.

Por estos motivos es por los que finalmente se decidió usar un software de renderizado en lugar de una librería para la generación de gráficos.

2.1.5. Renderizado por GPU y CPU

En la actualidad, la potencia de las GPUs está muy por encima de la de las CPU (*Central Processing Unit*)s, sin embargo, se diseñaron específicamente para realizar un trabajo concreto y eso hace que estén muy limitadas en cuanto a su uso en otros campos.

Las GPUs no suelen estar preparadas para trabajar con imágenes mayores que la resolución máxima que pueden proporcionar a una pantalla. Además,

la memoria de la que disponen, aunque trabaja a una velocidad mayor, también es más escasa. Es por eso que la mayoría de renderizadores dependen de la CPU únicamente, y las granjas de servidores utilizados por las grandes empresas a la hora de generar los fotogramas de las películas no se basan en el uso de GPUs de alta gama, sino en procesadores último modelo.

Aún así, el software de renderizado en la actualidad ya está investigando el uso de las GPUs como asistencia de la CPU a la hora de renderizar las imágenes, así que no sería de extrañar que en los próximos años esta situación sufriera un cambio radical. El software que se ha elegido para el proyecto ya tiene opción a utilizar la GPU en el renderizado, aunque aún es una opción en desarrollo que funciona únicamente con algunas tarjetas gráficas. Sin embargo, esto significa que en un futuro no muy lejano se podría hacer el cambio a renderizado por medio de las GPUs sin apenas cambios en el diseño del sistema global, proporcionando una mejora notable en los tiempos de realización de los trabajos de creación de imágenes.

2.1.6. CUDA y OpenCL

En un intento de aprovechar de otra manera el potente hardware que incluían las GPUs modernas, se crearon estos lenguajes que permiten realizar tareas a una mayor velocidad aprovechando la capacidad para el procesamiento en paralelo de la que disponen las GPUs actuales. Sin embargo, no todos los programas son sencillos de modificar para que funcionen en una GPU y la aprovechen al máximo, y es por eso que el uso de estos lenguajes no es muy común fuera de los ámbitos científicos, en los que se suelen utilizar para llevar a cabo simulaciones que exigen de la realización de muchos cálculos matemáticos fácilmente paralelizables.

Para este proyecto se descartó el uso de CUDA/OpenCL puesto que exigía disponer de GPUs modernas que solo iban a ser útiles a la hora de realizar algún procesado de los datos dada la situación actual de los renderizadores por GPU.

2.2. Big Data

Como se ha dicho con anterioridad, vivimos en una época en la que la tecnología nos rodea a todas horas, y la cantidad de datos que se puede obtener de las actividades que realizamos a diario es enorme. Durante los últimos años se ha hecho muy común el uso del término "Big Data" para referirse a estas cantidades enormes de datos que los consumidores y usuarios generan, y que están relacionados con sus gustos, sus costumbres, su mentalidad, etc. Este término también se utiliza para referirse a los sistemas que almacenan toda esa información, generada por los usuarios, y que se encargan de analizarla en busca de patrones que se repitan con una determinada frecuencia dentro de los datos. Estos patrones permiten hacer deducciones, e incluso predicciones, sobre la fuente, o el evento, que ha generado los datos.

Las mayores dificultades que afectan a esta disciplina son las de gestión, análisis y visualización de los datos. Es cierto que, el coste del almacenamiento de los datos, se ha abaratado mucho en los últimos años. Sin embargo, si se quiere obtener alguna ventaja de los datos obtenidos, se tienen que gestionar de una manera efectiva, y que sea lo más eficiente posible a futuro, puesto que el conjunto de datos continuará creciendo sin fin. Este es un problema al que se enfrentan los científicos, pues las cantidades de datos que se generan con los experimentos y simulaciones van en aumento. Los avances en la tecnología hacen que los instrumentos que se utilizan para medir los diferentes fenómenos, naturales o artificiales, que los científicos investigan, sean cada vez más precisos, y proporcionen cantidades de datos mayores.

Pero no solo afecta a los científicos el ámbito del "Big Data". A causa de la sociedad globalizada en la que vivimos, los economistas cada vez disponen de más información sobre los movimientos de las empresas de todo el mundo, y como es bien sabido, la información es poder. Un sistema capaz de analizar datos económicos obtenidos por todo el mundo puede buscar patrones que se repitan, y el dueño de ese sistema conocería las que, en ocasiones anteriores, han sido las consecuencias de esos patrones, y podría beneficiarse de ellas.

Hoy en día, la cantidad de datos generados por persona y unidad de tiempo es inmensa, y a futuro, seguirá en aumento. Es por eso que, actualmente, grandes organizaciones y empresas dedican muchos recursos al desarrollo de tecnologías para trabajar con lo que será el futuro de la información, el "Big Data".

2.2.1. NumPy y SciPy

Estas son algunas de las consecuencias de que Python sea un lenguaje muy utilizado en el ámbito científico y han causado que cada vez se use más en el campo del "Big Data".

- Numpy (http://www.numpy.org/)
  Es una librería que proporciona estructuras de datos para operar de una manera muy eficiente con arrays y matrices (arrays multidimensionales). Estas estructuras de datos permiten que el código en Python que utilice esta librería para operar con arrays, pueda ejecutarse casi a la misma velocidad que el mismo código programado en C, con la ventaja de la facilidad que implica el programar en Python en lugar de usar un lenguaje de más bajo nivel.

- SciPy (http://www.scipy.org/)
  Es una biblioteca que contiene módulos para facilitar la realización de tareas de optimización, álgebra lineal, integración, interpolación, procesamiento de señales y de imagen, resolución de ODEs y otras tareas de ciencia e ingeniería.

2.2.2. Pandas

Debido a este boom de popularidad del Big Data, han aparecido varias librerías para facilitar el tratamiento de datos. Una de las más conocidas es "Pandas" (http://pandas.pydata.org/), que proporciona herramientas y estructuras de datos para que podamos operar con grandes cantidades de información de una manera más sencilla y optimizada.

Pandas es una librería de alto nivel que se aprovecha de muchas de las ventajas que proporcionan las dos librerías vistas anteriormente (Numpy y SciPy). La existencia de esta librería ha hecho que Python sea uno de los lenguajes más utilizados en el ámbito del Big Data hoy en día.

2.3. Sistemas distribuidos

A la hora de ofrecer un servicio a los usuarios, hay que tener en cuenta que ni el flujo de usuarios, ni la carga de trabajo que estos demanden del sistema, va a ser constante a lo largo del tiempo de funcionamiento de tu servicio. Es aquí donde entra en juego la importancia de los sistemas distribuidos, muy populares en la actualidad.

Un sistema distribuido permite aprovechar varios computadores independientes conectados mediante la red para llevar a cabo un trabajo en común, y, en la mayoría de los casos, de una manera más eficiente de la que lo haría un solo computador con mayor capacidad de proceso, gracias a las ventajas que supone el poder paralelizar el trabajo .

En este proyecto, el renderizado de imágenes exige mucho trabajo al procesador, y si se utilizase un solo sistema para llevar a cabo la creación de las imágenes, se formaría una cola con los trabajos de todos los usuarios. El uso de un sistema distribuido permite proporcionar el servicio a una cantidad de usuarios variable sin que el trabajo que realiza un usuario pueda afectar a los demás.

Los computadores que forman parte del sistema distribuido (también llamados nodos) llevan a cabo distintos trabajos en función de su posición dentro del sistema. En una situación de poca carga de trabajo, en la que apenas hay usuarios que estén utilizando el sistema, una baja cantidad de nodos serviría para hacerlo funcionar correctamente. En el momento en que el sistema se viese sobrecargado, solo sería necesario añadir algunos nodos más al sistema para que ayuden con el trabajo en caso de que aumentase de manera repentina. La conexión y desconexión de nodos del sistema se puede llevar a cabo

en cualquier momento, y el sistema reaccionará automáticamente.

2.3.1. Comunicación entre nodos del sistema

En un sistema distribuido formado por distintos computadores, se necesita disponer de un sistema de intercambio de mensajes, de manera que los distintos nodos del sistema puedan ponerse de acuerdo en el reparto de trabajo y otras cuestiones a las que tienen que hacer frente de manera conjunta.

La manera más sencilla de solucionar este problema es utilizar un software que haga de servidor de mensajes, y al que cualquier nodo del sistema se pueda conectar para enviar y recibir mensajes.

A continuación se mostrarán dos de las opciones que se investigaron a la hora de decidir el modelo a utilizar para este proyecto:

- RabbitMQ (https://www.rabbitmq.com/)
  Es un software de negociación de mensajes de código abierto, que implementa el estándar AMQP (*Advanced Message Queuing Protocol*). Este software solo se encarga de la gestión de los mensajes, y hay disponibles librerías para muchos lenguajes de programación que permiten la interacción con el servidor para enviar y recibir los mensajes.

- Redis (http://redis.io/)
  Redis es una base de datos que se basa en almacenar tablas de hashes (pares clave/valor) en memoria, lo que la hace muy rápida a la hora de guardar y obtener la información que se almacena en ella. Aunque también permite su uso como una base de datos permanente, guardando la información en el disco cada cierto tiempo, se suele utilizar como una caché de acceso rápido entre un sistema y la base de datos final (que suele ser MySQL, MongoDB, etc.). Además dispone de un sistema de canales y suscriptores, que permite utilizarla como mecanismo de intercambio de mensajes.

En este proyecto se decidió usar la base de datos Redis porque ofrece algunas características adicionales al envío y recepción de mensajes que se han

utilizado para intercambiar información entre los nodos del sistema distribuido. Estas características, como son el almacenamiento de pares clave/valor o la posibilidad de asignar un determinado tiempo de vida a estos pares, permiten intercambiar información que no necesariamente tiene que ser recibida en tiempo real. Esto hace que no se sature el canal de mensajes principal al que todos los nodos están conectados a la espera de recibir órdenes.

2.4. Computación en la nube

Es un modelo de prestación de servicios de negocio y tecnología, que permite al usuario acceder a un catálogo de servicios estandarizados y responder con ellos a las necesidades de su propio negocio, de una forma flexible en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado.

Se ha hecho muy común durante los últimos años, y la cantidad de servicios que se ofrecen a través de la nube va en aumento.

Estos servicios se engloban en tres grandes modelos:

1. IaaS (*Infraestructure as a Service*)
   Es el nivel más bajo de abstracción de estos modelos de negocio. Se basa en la venta del uso de infraestructura, ya sea como almacenamiento o como capacidad de proceso, de manera que el usuario final pueda disponer de ella sin tener que preocuparse de los costes de mantenimiento o sustitución de los sistemas. En muchos casos también se la denomina 'Hardware como servicio'(HaaS), puesto que todas las decisiones sobre el software a utilizar dependen del usuario final que contrata el servicio.

   Dentro de este modelo de negocio, el ejemplo comercial más famoso es el de Amazon, que a través de sus servicios EC2 (*Elastic Cloud Computing*) y S3 (*Simple Storage Service*) proporciona a los usuarios la infraestructura para realizar trabajos de cómputo y almacenar datos.

2. PaaS (*Platform as a Service*)
   Es la capa media en cuanto a los modelos de negocio basados en la nube. Se ofrece un conjunto de complementos que permiten al usuario final hacer funcionar su servicio. En la mayoría de los casos estos complementos suelen ser servidores web, bases de datos, APIs de acceso a otros servicios, etc. que permiten al usuario final centrarse en el diseño e implementación de su servicio.

   Un ejemplo bastante conocido de este tipo de modelo de negocio es el del Google App Engine, que aprovecha la infraestructura de la que dispone Google para facilitar a los usuarios finales la opción de desarrollar y ejecutar aplicaciones basadas en diferentes lenguajes de programación y tecnologías.

3. SaaS (*Software as a Service*)
   Este es el nivel más alto de abstracción, y el que más desarrollo está teniendo en la actualidad. El usuario final dispone de un servicio a través de la nube que en la mayoría de los casos compite directamente con los servicios que podría obtener en su propio sistema. Estos servicios en la nube ofrecen ventajas a los usuarios, como son:

   - Acceso al servicio desde cualquier localización desde la que se disponga de acceso a la red.
   - El servicio es independiente del sistema desde el que se utilice.
   - Los datos del usuario son accesibles desde cualquier sistema.

   Algunos ejemplos muy conocidos de este modelo de negocio son: Gmail, Google Docs o Dropbox.

Este proyecto diseña un sistema que ofrece software como servicio (SaaS) a los usuarios finales. La infraestructura necesaria para hacer funcionar el servicio puede ser tanto privada como subcontratada a una empresa especializada (como Amazon). Esto último facilitaría el aumento, de una manera rápida, de la cantidad de nodos que forman el sistema en caso de picos de trabajo.

2.4.1. Infraestructura privada y pública

El uso de servicios en la nube ofrece muchas ventajas, pero también presenta algunos problemas que no se tienen cuando la infraestructura que se utiliza es privada.

Uno de los problemas que acarrea el uso de los servicios en la nube es que en muchos casos puede significar la pérdida del control sobre nuestros datos. Toda la información está presente en la infraestructura de la empresa que nos proporciona el servicio, o en algunos casos en la de otras empresas distintas que le proporcionan servicios a la primera.

Uno de los ejemplos más claros de esto es el caso de Dropbox. Esta empresa ofrece a los usuarios una manera sencilla de almacenar en la nube sus datos, y de tenerlos accesibles desde cualquier dispositivo que se conecte a la red. Podría pensarse que cuando subes tus archivos a Dropbox los estás enviando a sus servidores, sin embargo, esto no es del todo correcto. Esta empresa se encarga de ofrecer a los usuarios un software como servicio (SaaS) para que puedan almacenar sus datos de una manera fácil y rápida, pero el almacenamiento de esos datos que los usuarios les envían no está dentro del modelo de negocio de Dropbox. Dropbox a su vez contrata a Amazon sus servicios de almacenamiento en la nube (Amazon S3), de manera que no necesita preocuparse de gestionar y mantener la infraestructura necesaria para almacenar los datos. Este es un claro ejemplo de que al utilizar servicios en la nube no siempre se tiene la certeza de dónde acaba almacenada la información, y de cuáles y cuántas empresas tienen acceso a ella.

A esto hay que añadirle el hecho de que en algunos casos los servidores de las empresas que ofrecen estos servicios en la nube se encuentran en otros países distintos al de la empresa inicial, y estos países pueden tener leyes diferentes sobre la protección de datos o la posibilidad de que su gobierno acceda a la información que está allí almacenada.

Capítulo 3

Arquitectura del servicio

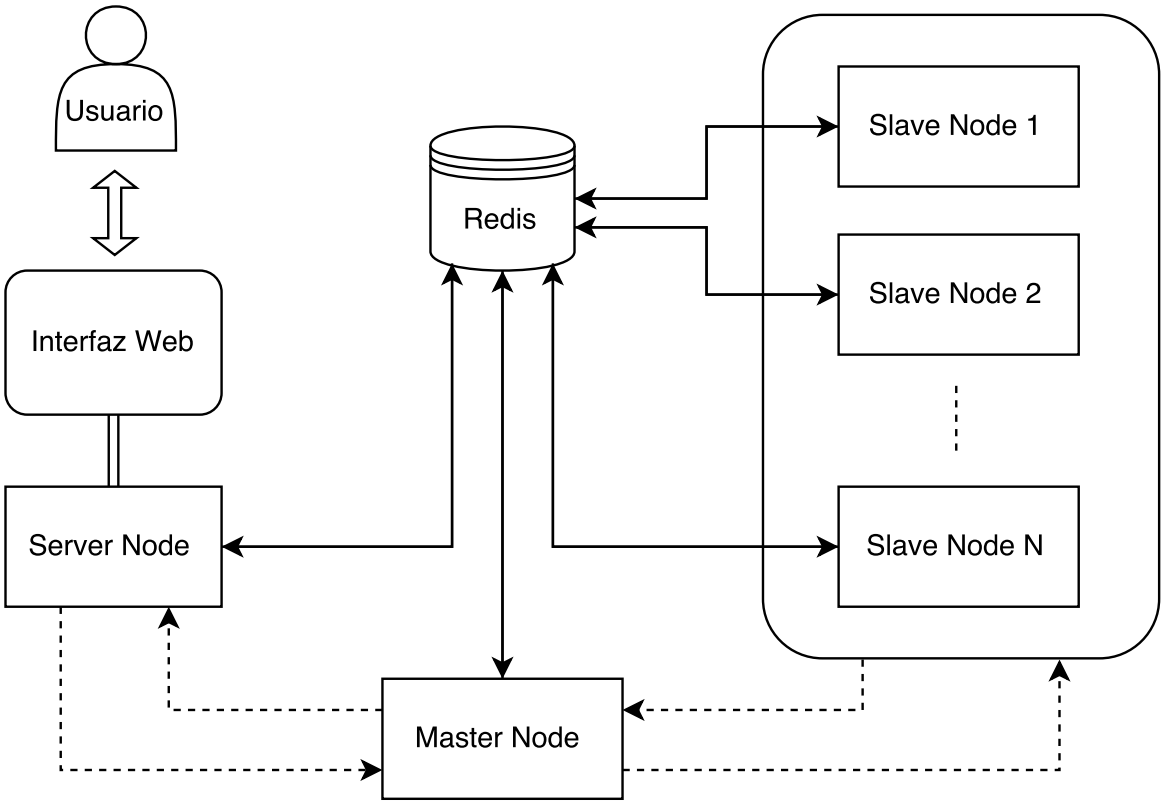
Resumen: En este capítulo se explica la arquitectura del sistema distribuido, los protocolos de comunicación que usan los nodos para intercambiar información entre ellos, y las tecnologías utilizadas en el desarrollo del sistema.

A la hora de diseñar esta aplicación se ha tenido en cuenta que la demanda del servicio por parte de los usuarios será variable, tanto por el número de usuarios que puedan usar el servicio en un momento concreto, como por el tipo de uso que le vaya a dar cada uno de ellos. Para evitar que los trabajos de unos pocos usuarios puedan saturar el sistema y afectar al resto de usuarios, se ha utilizado un sistema distribuido.

Además la aplicación se divide en 2 partes principales:

1. **Front-end**: el servidor web que interacciona con los usuarios.
   Proporciona una interfaz para enviar los datos a procesar y el acceso a las opciones generales de configuración para el renderizado de la visualización. Está en comunicación con el nodo maestro del sistema.

2. **Back-end**: conjunto de nodos del sistema.
   Son los encargados de realizar el trabajo final que el usuario requiera. La cantidad de nodos que forman esta parte del sistema es variable, y puede cambiar mientras el sistema se encuentra en funcionamiento.


Usuario
Interfaz Web
Server Node
Redis
Slave Node 1
Slave Node 2
Slave Node N
Master Node

3.1. Front-end

Se utiliza una página web para proporcionar al usuario final del servicio una interfaz a través de la cual pueda enviar los datos que quiere procesar, así como elegir entre diversas opciones que afectan al resultado final del trabajo. Esto permite abarcar un gran número de posibles clientes de nuestro servicio, puesto que lo único que necesitan ellos es disponer de un navegador web, presente en la gran mayoría de dispositivos que usamos hoy en día.

Las funciones de las que se encarga el servidor web son las siguientes:

- Recibir del usuario los datos necesarios para realizar el trabajo, ya sea a través del interfaz web, o mediante un archivo que el usuario proporcione.
- Enviar la orden de trabajo al nodo maestro con todos los datos necesarios para llevarla a cabo.
- Proporcionar al usuario final un enlace para obtener el resultado final en máxima calidad.

La aplicación funcionando en el servidor web utiliza un script externo para comunicarse con el nodo maestro a través de una instancia de la base de datos Redis (ver Base de datos en la sección 3.2.3). Este script se encarga de enviar las órdenes de trabajo a través de uno de los canales de la base de datos, para que el nodo maestro las procese y reparta su trabajo al sistema de nodos trabajadores.

3.1.1. Script “server_node.py”

Este es el script del que dispone el servidor web para interaccionar con el sistema distribuido. Su principal objetivo es enviar las órdenes de trabajo al sistema una vez que el usuario proporciona los datos y las diferentes opciones de configuración.

En una ejecución normal, el script sigue estos pasos:

1. Crear ID de trabajo.
   Se genera un identificador para el nuevo trabajo que se ha de realizar, y se almacena este identificador en la base de datos. También se almacena información sobre el estado del trabajo, o los archivos que lo componen.

2. Copia de los archivos.
   Los archivos necesarios para la correcta realización del trabajo se copian al nodo maestro de manera que estén disponibles para el nodo del sistema que tenga que trabajar con ellos.

3. Envío de la orden de trabajo.
   Se utiliza el canal "server" de la base de datos para enviar al nodo maestro un mensaje con la nueva orden de trabajo, y se espera la respuesta de confirmación de recepción.

4. Chequeo del estado del trabajo.
   Se comprueba periódicamente el estado del trabajo, o cuando el usuario lo desee, y en el momento que está disponible se ofrece al usuario.

3.1.2. Archivo de entrada de datos

El archivo que el servidor envía al sistema con los datos tiene un formato similar a CSV (*Comma-Separated Values*), aunque se le han añadido algunas opciones que permitan al usuario añadir, a los valores numéricos que forman los datos, determinados parámetros de configuración que se usen en la representación. Se diseñó este formato de datos de entrada porque los archivos CSV son muy comunes a la hora de importar y exportar datos desde diferentes programas. Por ejemplo, las hojas de Microsoft Excel son fácilmente transformables en archivos CSV. De esta manera el usuario puede exportar los datos desde otro programa a un archivo CSV y modificar este archivo añadiéndole las secciones necesarias para la representación de los datos de la forma elegida.

El usuario puede enviar el archivo de datos generado por su cuenta, o utilizar la interfaz que se le proporciona a través de la web para configurar las diferentes opciones.

El archivo se compone de 3 partes principales, la sección de configuración general (sección "CONFIG"), la sección donde se declaran variables para utilizarlas en la sección de los datos (sección "VARS"), y la sección de datos que se utilizará para generar los gráficos (sección "DATA"). La sección "VARS" no es necesaria, y se puede omitir si todo el gráfico se va a renderizar utilizando las mismas propiedades de color y material en el shader. Las secciones están delimitadas por dos líneas que tienen que contener el nombre de la sección (en mayúsculas o minúsculas) en el caso de el delimitador de inicio de la sección, y la palabra "END" (no distingue entre mayúsculas y minúsculas) en el delimitador de fin de cada sección.

3.1.2.1. Sección "CONFIG"

Esta sección, la de configuración general, estará formada por los siguientes parámetros (cada uno en su propia línea):

- image-size: Este parámetro indica las dimensiones finales de la imagen a generar. Va seguido de dos números enteros que representan el ancho y alto de la imagen respectivamente.
  Ej: `image-size; 1920; 1080`
- image-type: Indica el tipo de imagen a generar. Se pueden generar imágenes en los formatos JPG, PNG y BMP.
  Ej: `image-type; png`
- graph-type: Indica el tipo de gráfico a generar, por ahora puede contener un valor de "histogram" o "cake".
  Ej: `graph-type; histogram`
- default-material: Indica el material por defecto que se usará en el gráfico, más adelante se explica como declarar un material, y los tipos posibles.
  Ej: `default-material; material(glass, Beckmann, 0.2, 1.52)`

- default-color: Indica el color por defecto utilizado en el gráfico, más adelante se explica cómo se declaran los colores.
  Ej: `default-color; color(0.8, 0.3, 0.4)`

- floor-color: Indica el color del suelo del gráfico.
  Ej: `floor-color; color(0.9, 0.9, 0.9)`

- camera-type (opcional): Tipo de cámara a utilizar. Puede contener un valor de "perspective" (es el valor por defecto en caso de que no se indique) o "orthographic".
  Ej: `camera-type; perspective`

- camera-transform (opcional): Aplica una transformación a la cámara. Más adelante se explican las transformaciones posibles y la manera de declararlas.
  Ej: `camera-transform; translate(10, 2, -2)`

- background-strength (opcional): Un número que representa la intensidad de la iluminación ambiente que rodea el gráfico.
  Ej: `background-strength; 0.8`

Dependiendo del gráfico utilizado, se pueden añadir también algunos parámetros más en la configuración. Estos parámetros son todos opcionales.

- (Histogram) bar-width; Indica la anchura de las barras en el histograma.

- (Histogram) bar-separation: Dos números que indican la separación de las barras en el histograma en los ejex X y Z.

- (Histogram) minimum-bar-height: Indica la altura mínima de un barra en el histograma. Se utiliza para calcular la altura de las demás en proporción.

- (Cake) cake-height: Indica la altura del gráfico de pastel.

3.1.2.2. Sección "VARS"

La segunda sección del archivo permite declarar constantes, que se utilizarán después, entre los datos, para variar el color o los materiales de algunas partes del gráfico. Los nombres de las variables comienzan con el símbolo "$" que forma parte del nombre de la misma, y están delimitados por el carácter ";" (este último no forma parte del nombre de la variable). Las variables pueden almacenar un color o un material únicamente.

Los materiales disponibles en el sistema son los siguientes:

- Difuso: El material no deja pasar luz por su interior y la luz reflejada se expande en todas direcciones, creando reflejos difuminados sobre la superficie del objeto. A la hora de crearlo, se añade un número comprendido entre 0 y 1 que representa lo afiladas que serán las sombras en los bordes del objeto. Se recomienda un valor en torno a "0.2" para una correcta visualización de los bordes de los elementos. Para declararlo se utiliza la siguiente función:
  `material(diffuse, <roughness>)`
- Brillante: El material es muy similar al "Difuso" visto anteriormente, pero los reflejos de luz sobre el objeto son más definidos. Cuando se crea, se tienen que añadir el tipo de distribución a utilizar (que puede ser "Beckmann" o "Sharp"), un valor idéntico al del material anterior, y otro que representa el "índice de refracción" del material. Para declararlo se utiliza la siguiente función:
  `material(glossy, <distribution>, <roughness>, <ior>)`
- Cristal: El material permite que la luz pase por su interior. Los valores de la configuración son los mismos que los del material anterior. Para declararlo se utiliza la siguiente función:
  `material(glass, <distribution>, <roughness>, <ior>)`

Cuando se quiere declarar un color se necesitan proporcionar los valores decimales (comprendidos entre 0 y 1) de cada uno de los 3 canales estándar del formato RGB (*Red Green Blue*). Se utiliza la función de la siguiente manera: `color(<red>, <green>, <blue>)`.

Las transformaciones se pueden aplicar a la cámara para cambiar su posición y orientación. Existen dos tipos de transformaciones:

- Traslación: Varía la posición de la cámara, según los valores que el usuario ha porporcionado. Estos representan los ejes X, Y y Z en ese orden. Para declarar esta tranformación se utiliza la siguiente función: `translate(<x>, <y>, <z>)`
- Rotación: Gira la cámara sobre uno de los ejes de coordenadas. El usuario indica el ángulo de giro y el eje sobre el que se aplica. En los sistemas de coordenadas que se utilizan en "Cycles", se puede determinar la dirección de giro que se aplicará utilizando la "regla de la mano izquierda". Para declarar una tranformación de rotación, se utiliza la siguiente función:
  `rotate(<angle>, <x>, <y>, <z>)`

Las transformaciones se pueden apilar unas sobre otras. Hay que tener en cuenta que, los sistemas de ejes sobre los que se aplica una transformación pueden haberse visto modificados por alguna transformación previa. A continuación vemos un ejemplo en el que se aplica una traslación seguida de una rotación:

```
translate(0, 4, -2).rotate(20, 0, 1, 0)
```

3.1.2.3. Sección "DATA"

Esta sección es la más importante del archivo, ya que es donde se incluyen los datos que el usuario quiere que se conviertan en un gráfico para su visualización y mejor análisis. En esta sección, cada línea del archivo representa una fila de la matriz de datos, y dentro de esta, los diferentes datos están separados por un punto y coma. Un ejemplo básico de datos en formato CSV es el siguiente:

```
1; 2; 3; 4; 5; 6; 7; 8; 9
100; 200; 300; 400; 500; 600; 700; 800; 900
1; 1; 2; 2; 3; 3; 4; 4; 5
```

En la sección de datos se pueden utilizar las variables que se hayan definido previamente en la sección "VARS". Para utilizar las variables se escribe su identificador (con el símbolo $ incluido) seguido de el símbolo "$" para indicar donde comienza y termina el rango de datos a los que afecta esa variable.

En el siguiente ejemplo se muestra una variable (identificador `$red`) aplicada a toda una línea de datos en la primera línea, y una variable (identificador `$blue`) aplicada a un único elemento de la línea.

```
$red$ 1; 2; 3; 4; 5; 6 $
10; $blue$ 20 $; 30; 40; 50; 60
```

Las variables se pueden encadenar unas dentro de otras, pero en el caso de aplicar más de un color o material solo tendrá efecto el de rango más amplio, sobrescribiendo a los del mismo tipo que tenga en su interior.

3.1.2.4. Ejemplo de archivo de entrada de datos

A continuación se muestra un archivo de ejemplo con el formato que el usuario utiliza para proporcionar los datos al sistema. Como se ha dicho previamente, el usuario puede proporcionar este archivo al sistema directamente, o enviar los datos y elegir las diferentes opciones de configuración a través del interfaz web. En el caso de que se quieran usar variables para modificar determinadas partes del gráfico con precisión será necesario incluir esas secciones a mano.

```
CONFIG
image-size; 800; 600
image-type: png
graph-type; histogram

default-material; material(difusse, 0.3)
default-color; color(0.8, 1.0, 1.0)
floor-color; color(0.7, 0.7, 0.7)
background-strength: 0.8

bar-width; 1.0
bar-separation; 1.5, 1.5
END_CONFIG

VARS
$blue; color(0.0, 0.0, 1.0)
$glass; material(glass, Beckmann, 0.2, 1.52)
END_VARS

DATA
1; 2; 3; 4; $blue$5; 6; 7$; 8; 9; 10
$glass$11; 12; 13; 14; 15; 16; 17; 18; 19; 20
END_DATA
```

Las líneas en blanco, o las que comiencen con el símbolo "#" son ignoradas por el procesador, de manera que se puedan añadir comentarios al archivo y separar las líneas para facilitar su comprensión por parte del usuario. Aunque en este archivo de ejemplo la sección de datos es muy pequeña, no hay un límite en cuanto a la cantidad de líneas o la longitud de las mismas. Sin embargo el usuario deberá tener en cuenta que el tiempo que se tarda en procesar su trabajo y generar el gráfico depende de la cantidad de datos que vayan a formar parte de él.

3.1.3. Tecnologías utilizadas

- Bootstrap (http://getbootstrap.com/)
  Es un conjunto de archivos de código en Javascript y hojas de estilos CSS (*Cascading Style Sheets*) que permite desarrollar sitios web que se comporten correctamente en una amplia variedad de dispositivos y tamaños de pantalla.

- Flask (http://flask.pocoo.org/)
  Este es un microframework de desarrollo web que utiliza el lenguaje Python como base, y el motor de plantillas Jinja2. Para este proyecto se escogió este framework puesto que el lenguaje Python era el que se ha usado en el resto de partes del sistema, por sus ventajas a la hora de realizar prototipos de aplicaciones de todo tipo. El script que permite al servidor web enviar las órdenes de trabajo para que sean procesadas por el sistema está escrito en Python, pero no exige que el servidor web utilice este lenguaje. Por lo tanto, el servidor web puede estar implementado en cualquier tecnología web (PHP, NodeJS, etc.) mientras este utilice el script en Python para el envío de trabajos al sistema.

3.2. Back-end

Este es el conjunto de programas que se encargan de proporcionar a los usuarios el servicio final. Para facilitar la escalabilidad en un futuro dependiendo de la carga de trabajo a la que se enfrente, se ha diseñado un sistema distribuido basado en tres tipos de nodos:

1. Nodo organizador (maestro): es un nodo único, que se encarga de la interacción con el servidor web, y de organizar el trabajo de los nodos trabajadores.

2. Nodos trabajadores (esclavos): realizan los trabajos finales de renderizado, procesado de imágenes, análisis de datos, etc. a petición del nodo maestro.

3. Base de datos Redis: la utilizan otros nodos en las comunicaciones necesarias para el funcionamiento del sistema distribuido.

La ventaja que ofrecen estos 3 tipos de nodos es que son completamente independientes unos de otros. Puede haber varios nodos en una misma máquina, o repartirlos por diferentes localizaciones, siempre que tengan acceso entre ellos a través de la red. Todos los nodos utilizan un módulo que carga la configuración desde un archivo JSON (*JavaScript Object Notation*) en el momento de su inicialización. De esta manera, se pueden configurar los diferentes nodos del sistema sin necesidad de modificar variables dentro del código fuente de cada uno. En apéndice A se explican los parámetros de estos archivos de configuración, y se incluyen varios ejemplos.

3.2.1. Nodo organizador (Nodo maestro)

Este nodo actuará de intermediario entre el servidor web y los nodos esclavos que realizan casi todo el trabajo final. Entre sus funciones están las siguientes:

- Llevar el control de los nodos trabajadores conectados al sistema en cada momento, así como de sus especificaciones generales para poder

utilizarlos más eficientemente.

- Recibir las órdenes de trabajo del servidor web, y repartirlas entre los nodos presentes en el sistema que no se encuentren ya ocupados con algún trabajo.
- Gestionar el trabajo que está realizando cada uno de los nodos trabajadores que está conectado al sistema y unificarlo cuando sea necesario.
- Reorganizar los trabajos en caso de que algún nodo se desconecte del sistema debido a un fallo repentino.

La carga de trabajo de este nodo, en condiciones normales, no será muy alta. Por lo tanto es posible ejecutarlo en la misma máquina que haga funcionar el servidor web o la base de datos, facilitando además la comunicación entre ellos y el copiado de ficheros, y agilizando el sistema.

3.2.2. Nodo trabajador (Nodo esclavo)

Estos nodos son los que se encargan de la realización del trabajo final que solicita el usuario. El sistema necesita, al menos, de un nodo trabajador para poder suministrar el servicio al usuario final. Este tipo de nodo puede funcionar en servidores diferentes al del nodo maestro, siempre que estén conectados a través de la red. En una situación ideal, un nodo trabajador no debería ejecutarse en la misma máquina que el nodo maestro, la base de datos o el servidor web, para evitar que la realización de los trabajos de los que se encarga este pudiesen afectar a los demás usuarios del sistema de cualquier manera.

En el arranque los nodos trabajadores generan un identificador, comprueban que no está en uso por otro nodo del sistema, y lo almacenan en la base de datos. Una vez comprobada la comunicación con el nodo maestro, realizan un análisis básico de sus especificaciones y de la carga de trabajo presente en el sistema en el que se están ejecutando. Esta información se almacena en la base de datos de manera que el nodo maestro pueda utilizarla a la hora de tomar las decisiones sobre el reparto del trabajo.

Una vez finalizado el proceso de inicio, el nodo trabajador habrá almacenado las siguientes claves en la base de datos en función de su identificador de nodo autogenerado:

```
NODE:<id>
NODE:<id>:nproc
NODE:<id>:status
NODE:<id>:job_id
```

Desde ese momento se mantendrán a la espera de recibir órdenes, comprobando periódicamente su conexión con el sistema del nodo maestro.

La comunicación entre los nodos maestro y esclavos es continua, independientemente de que estén realizando algún tipo de trabajo o estén a la espera. Esto permite que el sistema pueda reaccionar rápidamente en el caso de que un nodo pierda la conexión, redirigiendo sus trabajos (si los hubiese) a otro nodo disponible, para afectar a los usuarios en la menor medida posible.

En caso de fallos de comunicación, el nodo intentará realizar chequeos de la red, para comprobar si el fallo es suyo, si ha afectado al nodo maestro, o a varios nodos del sistema (estos chequeos solo serán viables en sistemas en los que la base de datos se encuentre en un servidor diferente al del nodo maestro). En esta situación, los nodos trabajadores están diseñados para continuar con sus tareas en el caso de que tuviesen alguna asignada. Independientemente de que el nodo maestro pudiese haber redirigido su tarea a otro de los nodos trabajadores, esta se finalizará y el resultado final se almacenará en el sistema hasta que se reciban órdenes desde el nodo maestro de cómo actuar.

Los nodos trabajadores tienen a su disposición varios scripts para realizar los diferentes trabajos, como son los siguientes:

- `renderer.py`
  Se encarga de ejecutar el programa externo que se usa de renderizador, con los archivos finales necesarios para que realice su trabajo. Dispone de varias opciones de configuración que obtiene del archivo

`renderer.cfg`. Este script necesita que en el sistema esté disponible un ejecutable de "Cycles" compilado para esa arquitectura, y con las dependencias instaladas. En el Apéndice 1, sección A.3.2 se explica con más detalle.

- `graph_creator.py`
  Es el programa principal utilizado a la hora de generar los archivos XML (*eXtensible Markup Language*) con la información de los objetos que forman el gráfico a realizar. Analiza los archivos que contienen los datos del usuario y genera diversos archivos XML siguiendo el formato que utiliza el renderizador "Cycles".

3.2.3. Base de datos Redis

Para la comunicación entre el servidor web, el nodo maestro, y los nodos trabajadores se utiliza una base de datos Redis (ver Protocolo de comunicación en la sección 3.3). Aprovechando los mecanismos de publicación de mensajes y suscripción a canales que tiene esta base de datos, cualquier nodo puede realizar comunicaciones globales a todo el resto de nodos que formen parte del sistema en un momento dado.

Se ha utilizado esta base de datos porque su mecanismo de canales y mensajes puede realizar la labor que en otro caso tendría que hacer un software que actuase como servidor de mensajes (usando protocolos como el AMQP), pero además permite su uso como una base de datos de hashes. Esto permite que los nodos intercambien información sobre los trabajos a realizar, o su propio estado, almacenando esos datos en claves de la base de datos. De esta manera, el canal de mensajes se libera de determinada información que no es necesario que se procese en tiempo real, y que solo necesita ser accedida por el nodo encargado del trabajo correspondiente.

Además, Redis permite el uso de tiempo de vida en las claves de la base de datos, una funcionalidad que se ha utilizado para controlar de una manera sencilla el estado de conectividad de los nodos del sistema. En lugar de mantener un canal por el que los nodos estén continuamente enviando mensajes

para informar de que siguen a la espera de recibir mensajes, se ha optado por un mecanismo mucho más sencillo.

Como se ha dicho anteriormente, una de las claves que los nodos almacenan en la base de datos tiene el nombre `NODE:<id>:status`. Esta clave tiene un tiempo de vida de 3 segundos, y es reescrita por los nodos cada 2 segundos. De esta manera, poco después de que un nodo pierda la conexión al sistema de mensajes su clave se eliminará, y el nodo maestro sabrá que el nodo no está disponible sin necesidad de perder tiempo enviando mensajes a un nodo que muy probablemente no es capaz de recibirlos.

En el caso del envío de las órdenes de trabajo entre los nodos del sistema, también se utiliza Redis. En este caso, las claves que se almacenan en la base de datos para cada trabajo dependen de su identificador, y tienen los siguientes nombre:

```
JOB:<id>
JOB:<id>:state
JOB:<id>:error
JOB:<id>:type
JOB:<id>:input
JOB:<id>:input_size
JOB:<id>:output
JOB:<id>:samples
JOB:<id>:resolution
JOB:<id>:worker_id
```

La clave `JOB:<id>:state` puede almacenar uno de los 4 estados siguientes: WIP (Work In Progress), WAITING, DONE o ERROR. En este último caso, en la clave `JOB:<id>:error` se almacena una cadena de texto que se le debe mostrar al usuario informándole del motivo del error, para que lo subsane. En caso de que el error se deba a un fallo interno en el funcionamiento del sistema, este campo estará vacío.

La clave `JOB:<id>:type` indica el tipo de trabajo a realizar y almacena uno de los valores siguientes:

- RENDER, para trabajos de renderizado.
- GRAPH cuando el nodo se tiene que encargar de crear los archivos XML que forman el gráfico final.

Las claves con nombres "input" y "output" contienen arrays con los nombres de los ficheros de entrada y salida de cada trabajo. Por último en `JOB:<id>:worker_id` está disponible el identificador del nodo que está realizando dicho trabajo. Las claves `JOB:<id>:input_size`, `JOB:<id>:samples` y `JOB:<id>:resolution` permiten al nodo maestro tomar la decisión de a que nodo le envia el trabajo en función de las necesidades que pueda requerir, sin tener que procesar él mismo el fichero de entrada para obtener esa información.

La base de datos Redis se almacena en la memoria RAM de la máquina que la hace funcionar, lo que hace que las operaciones de lectura y escritura de datos sean muy rápidas. Tiene además opciones de guardar su estado en disco cada determinado tiempo, configurable en función de las claves que se hayan añadido o modificado desde el último punto de guardado. Esto permitiría recuperarse de una caída del servidor que mantiene la base de datos, pero es casi seguro que se producirá perdida de información en este suceso. Redis permite usar varias instancias de sus bases de datos en servidores distintos de manera redundante, permitiendo recuperar sin apenas fallos el sistema principal en caso de caída.

La información que los nodos almacenan en la base de datos es útil únicamente mientras el sistema está en funcionamiento, y no necesita ser guardada en disco una vez que se realice el apagado del sistema. El nodo maestro se encarga de hacer una limpieza de la base de datos previo a una desconecxión del sistema, de manera que no se deje información inútil en la misma que pueda ocupar espacio en la RAM o en disco.

3.2.4. Tecnologías utilizadas

A la hora de implementar los distintos nodos que forman parte del sistema, se ha utilizado Python y varias de las librerías que forman parte del paquete estándar de este lenguaje.

Los nodos utilizan el módulo "multiprocessing" para poder ejecutar varios procesos que aprovechen los diferentes procesadores presentes en el sistema (en caso de que tenga más de uno). Mientras el proceso inicial se encarga de las tareas principales que ha de llevar a cabo el nodo y ejecuta el resto de programas que el nodo utiliza en su trabajo, otro proceso monitoriza los canales de comunicación con el resto de nodos del sistema, a la espera de mensajes dirigidos a este nodo. Se planteó el utilizar la librería "Twisted" (https://twistedmatrix.com/trac/) para la implementación de los nodos, ya que está especialmente diseñada para desarrollar aplicaciones dirigidas por eventos y en la mayoría de los casos nuestros nodos reaccionan en función de los mensajes que reciben. Sin embargo es demasiado compleja para el uso que se le daría en este proyecto, y además sería una dependencia más a tener instalada en el sistema en el que se quisiese ejecutar el nodo, a diferencia de los módulos estándar de Python que ya están instalados en la gran mayoría de sistemas Linux.

Para las comunicaciones con la base de datos se utiliza "redis-py" que ofrece acceso a casi todos los comandos de las bases de datos Redis de un modo muy sencillo de implementar en Python.

Puesto que las conexiones a la base de datos Redis usando esta librería no están cifradas (ni siquiera a la hora de enviar la contraseña de acceso en caso de usarla), y su nivel de seguridad no es muy grande, no es recomendable exponer la base de datos al exterior. Esta debería estar protegida por un firewall que bloquease el acceso a la misma desde fuera de la máquina en la que se ejecuta. Para que los nodos puedan acceder a la misma evitando el firewall y, de paso, añadiendo cifrado a la conexión para poder utilizarla a través de redes que puedan estar comprometidas, se utiliza un túnel SSH (*Secure Shell*) entre la máquina del nodo y la máquina de la base de datos.

Para facilitar la creación de este túnel SSH existe el módulo `tunneler.py`, utilizado por todos los nodos del sistema salvo el de la base de datos. Este script se encarga de la creación del túnel siempre que sea necesario, basándose en los siguientes parámetros de la configuración del nodo:

- db_host: El hostname o la dirección IP de la máquina donde funciona la base de datos.
- db_user: El nombre de usuario utilizado por el nodo en la máquina.
- db_ssh_port: El puerto en el que funciona el servicio de SSH en esa máquina.
- db_redis_port: El puerto en el que funciona la base de datos en esa máquina.

Otro módulo presente en casi todos los nodos, en este caso en el nodo servidor y en todos los nodos trabajadores, es `file_mover.py`. Este script facilita el tener que mover los archivos hacia y desde el nodo maestro. Utiliza los siguientes parámetros de la configuración del nodo:

- master_host: El hostname o la dirección IP de la máquina donde se encuentra el nodo maestro.
- master_ssh_port: El puerto en el que funciona el servicio de SSH en esa máquina.
- master_dir: El directorio raíz donde el nodo maestro almacena los archivos de los trabajos.
- home_dir: El directorio raíz del nodo que ejecuta este script.

Para la creación de los túneles SSH y la copia de archivos en remoto utilizando SCP (*Secure Copy*), es necesario que tanto los nodos trabajadores como el script del servidor web tengan acceso por SSH a la máquina del nodo maestro. Este acceso deberá ser mediante claves públicas, para evitar que el servidor de destino solicite la contraseña a los scripts cuando se ejecuten

automáticamente. Se recomienda crear en todas las máquinas del sistema un usuario común (p.ej: nodo-render), que facilite la instalación y funcionamiento del sistema (ver Instalación y mantenimiento del sistema en el apéndice A).

Para crear las imágenes se utiliza “Cycles”, el motor de renderizado de “Blender”. Existe un programa independiente que proporciona el motor de renderizado únicamente, y que tendrá que estar disponible en todos los nodos del sistema que quieran hacer trabajos de renderizado. En un principio, el nodo maestro envía una orden de trabajo a un nodo trabajador para que procese la información que el usuario ha introducido en el sistema. Tras el procesado de estos datos se generan varios archivos XML que contienen la información necesaria para que el renderizador pueda crear la imagen final.

El primero de estos archivos contiene la configuración general a usar en el renderizador, con parámetros que indican las dimensiones de la imagen, la posición de la cámara, los objetos a renderizar con sus posiciones, las configuraciones de los shaders a utilizar, etc. El resto de archivos XML contienen los datos de los diferentes objetos que forman parte de la imagen final, y que irán agrupados en archivos en función de los shaders que se utilicen a la hora de renderizar cada conjunto de objetos.

3.3. Protocolos de comunicación

Como se ha dicho previamente, los diferentes nodos del sistema se comunican mediante el uso de los mecanismos de canales, suscripción y publicación que proporciona la base de datos Redis. A continuación se explican los dos protocolos diseñados, muy similares ambos, que permiten a los distintos nodos intercambiar información para el funcionamiento del sistema.

3.3.1. Entre el nodo servidor y el nodo maestro

Este sistema se basa en un canal "server" al que el nodo servidor envía los mensajes con las órdenes de trabajo que se generan a partir de la información que proporcionan los clientes. El nodo maestro se conecta a ese mismo canal y se mantiene a la espera de recibir mensajes con órdenes de trabajo, que debe procesar y repartir entre los nodos trabajadores del sistema.

Los mensajes tienen el siguiente formato general, basado en JSON:

```
{
    SRC         // identificador del emisor del mensaje
    TIME        // fecha del envío

    CMD         // código hexadecimal del comando
    RET         // código decimal adicional
    JOB_ID      // identificador del trabajo

    EXTRA       // diccionario adicional
}
```

El campo `SRC` contiene el identificador del nodo que ha enviado el mensaje. Esto permite evitar que un nodo procese y actúe en función de los mensajes que el mismo ha enviado. El campo `TIME` incluye la fecha y hora del momento de envío del mensaje, en tiempo Unix. El campo `CMD` es un valor hexadecimal que identifica el comando enviado. Algunos comandos utilizan además los campos `RET`, `JOB_ID` y `EXTRA` para añadir más información en caso necesario.

A continuación se explican con más detalle los comandos utilizados para la comunicación entre el nodo servidor y el nodo maestro:

- **Estado de la comunicación** `[COMM_STATUS]`

CMD	0x300

  Lo envía el nodo servidor antes de cada orden de trabajo, y espera que el nodo maestro conteste con un mensaje de `COMM_OK`. En caso negativo el nodo servidor realizará un par de intentos más, después de los cuales informará al usuario de que no es posible realizar el trabajo en ese momento debido a fallos en la conexión de red.

- **Comunicación en curso** `[COMM_OK]`

CMD	0x301
RET	...

  Este mensaje es la respuesta del nodo maestro a los mensajes preguntando por el estado de la comunicación que envía el nodo servidor. Esta respuesta se envía tan pronto como se recibe el otro mensaje, y en el campo `RET` lleva un número positivo que indica el número de milisegundos que tardó el mensaje inicial en ser recibido por el nodo maestro. Este valor puede utilizarse como una medida del estado de la comunicación.

- **Orden de trabajo** `[JOB_ORDER]`

CMD	0x200
JOB_ID	...

  El nodo servidor envía al nodo maestro un trabajo a realizar. El campo `JOB_ID` incluye el identificador del trabajo, y le sirve al nodo maestro para obtener de la base de datos la información relativa al mismo. Se envía una vez el nodo servidor ha copiado en el nodo maestro los archivos necesarios para la realización de dicho trabajo.

- **Confirmación de mensaje de trabajo** `[JOB_ACK]`

CMD	0x201
JOB_ID	...

El nodo maestro avisa de que ha recibido el mensaje sobre el trabajo al que se refiere el identificador del campo `JOB_ID`. En caso de no recibir una respuesta, el nodo servidor enviaría la orden un par de veces más con un retardo de tiempo ascendente. En caso de seguir sin obtener respuesta por parte del nodo maestro, informaría al usuario del error.

- **Rechazo de orden de trabajo** `[JOB_REJECT]`

CMD	0x202
JOB_ID	...
RET	...

El nodo maestro rechaza el trabajo con el identificador `JOB_ID` que le ha enviado el nodo servidor. El código almacenado en el campo `RET` indica el fallo que ha llevado al nodo a rechazar el trabajo.

  - RET = 0: El nodo maestro no dispone de nodos trabajadores a su cargo.
  - RET = 1: La conexión con los nodos trabajadores se ha perdido por causas desconocidas.

Antes de enviar una orden de trabajo al nodo maestro, el servidor crea un identificador de trabajo (id), y lo utiliza para añadir a la base de datos determinada información sobre el trabajo a realizar. Desde ese momento se mantiene a la espera de una respuesta de finalización de su trabajo, o de la causa del error que ha impedido realizarlo. Mediante el uso de AJAX (*Asynchronous JavaScript And XML*), el servidor web chequea un campo en la base de datos con el nombre `JOB:<id>:state` en el que se almacena uno de los siguientes valores:

- WIP: El trabajo está en proceso. Informa al usuario de esto y mantente a la espera.
- WAITING: El trabajo se encuentra en cola, y será procesado por el sistema tan pronto como sea posible.

- DONE: El trabajo ha sido finalizado, y está disponible para mostrárselo al usuario.
- ERROR: Ha habido un error con el trabajo por el que no se ha podido procesar correctamente.

En caso de error, en el campo `JOB:<id>:error` se encuentra una descripción del error que se debe mostrar al usuario para que lo solucione.

3.3.2. Entre el nodo maestro y los nodos trabajadores

El sistema se basa en un canal "workers" al que se suscriben tanto el nodo maestro como los nodos trabajadores, y mediante el cual se distribuyen mensajes que deberán ser procesados en tiempo real.

Estos mensajes tienen el siguiente formato general basado en JSON:

```
{
    SRC         // identificador del emisor del mensaje
    DST         // identificadores de los destinatarios
    TIME        // fecha del envío

    CMD         // código hexadecimal del comando
    RET         // código decimal adicional
    JOB_ID      // identificador de trabajo

    NODE_ID     // identificador de nodo (utilizado en los
                       mensajes de estado de la red)
}
```

El campo `SRC` contiene únicamente la identidad del nodo que ha enviado el mensaje. Esto nos permite evitar que un nodo procese y actúe en función de los mensajes que el mismo ha enviado.

El campo `DST` indica a quien va dirigido el mensaje, y puede contener 3 valores distintos:

- Una única identidad de nodo
- Un array con identificadores de los nodos a los que va dirigido el mensaje
- Valor nulo, que indica que el mensaje va dirigido a todos los nodos del sistema.

La mayoría de los mensajes irán dirigidos a un único nodo. Sin embargo la posibilidad de enviar mensajes a todos los nodos del sistema se utiliza en algunos casos cuando se han detectado fallos en el funcionamiento del sistema y se quiere hacer un chequeo de la conectividad.

El campo `TIME` incluye la fecha y hora del momento de envío del mensaje, en tiempo Unix.

El campo `CMD` es un valor hexadecimal que identifica el comando enviado. Determinados comandos utilizan además los campos `RET`, `JOB_ID` y `NODE_ID` para añadir más información.

A continuación se explican con más detalle los comandos utilizados por el protocolo:

- **Conexión** `[CONNECT]`

CMD	0x101

  Lo envían los nodos trabajadores al conectarse a la red. Antes del envío del mensaje, el nodo trabajador debe generar aleatoriamente una cadena de identificación, comprobar que no está en uso por parte de otro nodo del sistema y almacenarla para su uso. A continuación realizará una comprobación de las especificaciones propias (chequeo del número de procesadores de los que dispone en esa máquina) y enviará esta información a la base de datos del sistema. Después de estos dos pasos, enviará el mensaje CONNECTED a través del canal principal de manera que el nodo maestro sepa que hay un nuevo nodo disponible para la realización de trabajos.

- **Desconexión** `[DISCONNECT]`

CMD	0x120
RET	...

Lo envían los nodos trabajadores para informar de su desconexión de la red de trabajo. El comando incluye un código, almacenado en el campo `RET`, que indica el motivo de la desconexión, de manera que el nodo maestro pueda actuar en consecuencia:

  - RET = 0: Reinicio del sistema del nodo. El nodo volverá a estar disponible en un breve espacio de tiempo.
  - RET = 1: Apagado del sistema del nodo.
  - RET = 2: Alta carga de trabajo en el sistema del nodo.

- **Desconexión permitida** `[DISCONN_ALLOW]`

CMD	0x121

Lo envía el nodo maestro para avisar al nodo trabajador de que su desconexión ha sido recibida, y que puede realizarla sin problema.

- **Desconexión no permitida** `[DISCONN_DISALLOW]`

CMD	0x122

Lo enviará el nodo maestro para avisar a un nodo trabajador de que su desconexión es perjudicial para el sistema, y que debería evitarse siempre que sea posible. Dependiendo de la manera en que esté configurado el nodo trabajador, este mensaje podría ser inútil.

- **Orden de trabajo** `[JOB_ORDER]`

CMD	0x200
JOB_ID	...

El nodo maestro informa a un nodo trabajador de que se le ha asignado un trabajo a realizar. El campo `JOB_ID` incluye el identificador del trabajo, y le sirve al nodo trabajador para obtener de la base de datos la información relativa al mismo (tipo de trabajo, nombres de ficheros, etc.).

- **Confirmación de mensaje de trabajo** `[JOB_ACK]`

CMD	0x201
JOB_ID	...

El nodo avisa de que ha recibido el mensaje sobre el trabajo al que se refiere el identificador del campo `JOB_ID`, y que empieza a trabajar en ello. Permite que el nodo maestro tome medidas en caso de que el receptor de su mensaje no responda por el motivo que sea, de una manera más o menos rápida.

- **Rechazo de mensaje de trabajo** `[JOB_REJECT]`

CMD	0x202
JOB_ID	...

El nodo trabajador rechaza el trabajo que le han asignado con el identificador `JOB_ID` porque actualmente ya está ocupado con otro trabajo.

- **Trabajo finalizado** `[JOB_FINISHED]`

CMD	0x203
JOB_ID	...

El nodo trabajador avisa al nodo maestro de que el trabajo se ha completado con éxito. Antes del envío de este mensaje, el nodo trabajador copia los archivos con los resultados al nodo maestro y actualiza la información referida a este trabajo en la base de datos, en este orden.

- **Error en el trabajo** `[JOB_ERROR]`

CMD	0x209
RET	...
JOB_ID	...

El nodo avisa de que ha habido algún tipo de error en la realización del trabajo cuyo identificador se incluye en `JOB_ID`. El valor numérico del campo `RET` indica el tipo de error:

  - RET = 0: Reinicio/apagado inminente. El nodo ha recibido una notificación de reinicio/apagado durante la realización del trabajo.
  - RET = 101: Información incompleta. La información en la base de datos sobre el trabajo a realizar es incompleta.

- RET = 201: Archivo no encontrado.
- RET = 202: Archivo corrupto.
- RET = 301: Comando no disponible. El nodo no puede utilizar el comando necesario para realizar el trabajo.
- RET = 999: Error desconocido.

Este tipo de comando es utilizado tanto por los nodos trabajadores para informar al maestro de que no pueden realizar su trabajo a causa de un error, como por el nodo maestro para informar a otros nodos de que el resultado de su trabajo no es correcto o no se ha recibido.

- **Estado de la conexión** `[CONN_STATUS]`

CMD	0x300
NODE_ID	...

Utilizado por los nodos para preguntar por el estado de la conexión con otros nodos. En el campo `NODE_ID` se incluye el identificador del nodo sobre el que se pregunta el estado.

- **Conexión correcta** `[CONN_OK]`

CMD	0x301
NODE_ID	...

Respuesta de estado de la conexión. La conexión sigue establecida con el nodo indicado en el campo `NODE_ID`, y se han recibido mensajes que provienen de él con posterioridad al mensaje `CONN_STATUS` al que se responde.

- **Conexión perdida** `[CONN_LOST]`

CMD	0x305
NODE_ID	...

Respuesta de estado de la conexión. El nodo identificado por el campo `NODE_ID`, sobre el que se preguntaba en el mensaje `CONN_STATUS` previo lleva un tiempo sin enviar mensajes al sistema.

- **Conexión reestablecida** `[CONN_RECOVER]`

CMD	0x302
NODE_ID	...

Respuesta de estado de la conexión. El nodo identificado por el campo `NODE_ID`, sobre el que preguntaba el mensaje `CONN_STATUS` previo ha recuperado la conexión de red con el sistema.

Capítulo 4

Trabajo futuro en el proyecto

Resumen: En este capítulo se enumeran varias opciones posibles a la hora de continuar a futuro con el trabajo iniciado por este proyecto.

4.1. Renderizado de las imágenes

4.1.1. Renderizado distribuido

Por ahora, la creación de cada imagen se realiza al completo en un único nodo del sistema, puesto que el programa de renderizado que se utiliza aún no permite distribuir un mismo trabajo entre varios nodos. En un futuro es probable que el programa permita hacerlo, pero no se dispone de fechas.

Una posible solución sería la de colocar objetos planos delante de la cámara, emitiendo luz hacia ella justo a la distancia mínima para que aparezcan en la imágen, pero suficientemente lejos de los objetos que forman el gráfico para que no les afecten. De esta manera, y gracias a que los renderizadores determinan las caras visibles de los objetos antes de renderizarlos, se podría repartir el trabajo de una imágen entre varios nodos. Se enviaría el mismo archivo del gráfico a renderizar a varios nodos, pero a cada uno se le ocultaria una zona distinta de la imágen por medio de estos planos que tapan

trozos de la cámara. Después otro nodo se encargaría de cortar las imágenes y componer una a partir de los diferentes trozos creados previamente.

De esta manera se podrían renderizar imágenes muy grandes repartiendo ese trabajo entre varios nodos, lo que lo haría mucho más interesante para el usuario final del servicio.

4.2. Nodo maestro

4.2.1. Algoritmo de eficiencia para el reparto de los trabajos

El nodo maestro está programado de manera que decida a quién envía las órdenes de trabajo que provienen del nodo servidor.

Los distintos nodos del sistema pueden disponer de diferentes programas para la realización de tareas, en función de sus características propias. Por ejemplo, en nodos con poca capacidad de proceso se podría configurar que solo realizasen tareas de análisis de ficheros y no se encargasen de ningún tipo de renderizado, puesto que este bloquearía la CPU durante demasiado tiempo. El nodo maestro decide actualmente a quién envía las órdenes basándose simplemente en si el nodo puede hacer el trabajo y no está ocupado.

A futuro se podría mejorar el nodo maestro de manera que realizase cálculos sobre la optimización de las tareas, de tal forma que las enviase a los nodos en función de la que considerase que fuese la mejor estrategia para disminuir la carga de trabajo del sistema durante los próximos minutos de funcionamiento del mismo.

4.2.2. Encendido y apagado automático de nodos trabajadores en función de la carga de trabajo

El sistema está diseñado para funcionar sin un número fijo de nodos trabajadores, y se pueden conectar y desconectar nodos a este mientras está en ejecución sin que haya ningún problema. Sin embargo estos nodos tienen que ser iniciados por algo externo al sistema, ya sea una persona, una tarea de CRON, u otro script ejecutado automáticamente por el sistema. A futuro se podría mejorar el nodo maestro para que se encargase de añadir o quitar nodos dinámicamente al sistema distribuido, en función de la carga de trabajo y las predicciones que hiciese de la misma.

En el caso de que se vendiese este servicio a los usuarios finales, este sistema dinámico de conexión y desconexión de nodos mejoraría mucho los beneficios, pues evitaría mantener nodos encendidos y conectados al sistema en estado de espera en situaciones en las que la carga de trabajo es muy baja simplemente para solventar los problemas que podría causar un pico de trabajo.

4.3. Nodo trabajador

4.3.1. Ampliación de las visualizaciones posibles de los datos

Por el momento en el sistema solo se han implementado dos visualizaciones de datos distintas. A futuro habría que ampliar estas posibilidades con nuevas opciones para los usuarios. Algunos ejemplos posibles serían:

- Mapa de plano
  Una representación gráfica en 3D de una o varias superficies en las que se podrían utilizar diferentes colores en función de la altura, como se hace con los gráficos de barras.

- Proyecciones de la Tierra
  Una representación en 3D de diferentes proyecciones del mapa del mun-

do, de manera que los diferentes continentes o países puedan utilizarse como barras de un gráfico de alturas.

4.4. Interfaz web

4.4.1. Cuentas de usuario

El sistema permitiría a los usuarios mandar un trabajo a ejecutarse al mismo, y volver más tarde para comprobar el resultado. Los usuarios dispondrían una sección privada con la información del estado de los trabajos que han ingresado en el sistema, y enlaces a los archivos de resultado de los mismos.

4.4.2. Implementación de un sistema de pago en función de los servidores a utilizar

El sistema funcionaría mediante créditos que los usuarios tendrían que comprar. Estos créditos les permitirían el uso de diferentes tipos de servidores, más o menos potentes en función del coste en créditos de los mismos, para la realización de sus trabajos. Así, el coste final para el usuario lo decide él mismo en función de las características de su trabajo, y la rapidez con la que quiera disponer del resultado.

Apéndice A

Instalación

Resumen: En este capítulo se explican los detalles técnicos para la puesta en marcha de las distintas partes del sistema, y consejos sobre su funcionamiento.

A.1. Servidor Web

Para este prototipo se ha utilizado Flask, un framework de Python, para desarrollar la interfaz web que permite al usuario interaccionar con el sistema. Para hacer funcionar esta interfaz web, es necesario tener disponible un servidor Apache (https://httpd.apache.org/) en la máquina que se quiera usar como nodo servidor. Llevar a cabo esta instalación es muy sencillo, pero depende de la distribución de Linux que se utilice en el sistema del nodo.

Una vez tenemos el servidor Apache funcionando, hay que confirmar que está instalado `mod_wsgi`, una modificación que permitirá a Apache responder a las peticiones web que le lleguen utilizando Python y Flask de intermediarios. En un sistema basado en Debia, se puede instalar este mod con los siguientes comandos:

```
$ sudo aptitude update
$ sudo aptitude install libapache2-mod-wsgi python-dev
```

Una vez se han instalado todas las dependencias, habrá que activar este mod, con el siguiente comando:

```
$ sudo a2enmod wsgi
```

En la mayoría de los sistemas este comando se encarga de reiniciar el servidor Apache pero, en caso de que no lo haga, sería necesario reiniciarlo manualmente con el siguiente comando:

```
$ sudo service apache2 restart
```

Una vez se ha instalado `mod_wsgi` en el servidor, hay que copiar los archivos de la web a una carpeta desde la que este pueda acceder a ellos. Normalmente, en los sistemas Linux, las diferentes web que tiene que gestionar el servidor Apache estarán en el directorio `/var/www`, aunque también se podría configurar para que todos los archivos de la web estuviesen dentro del directorio `home` del usuario que ejecutará el nodo. Los archivos de la web incluyen un "ambiente" de Python con los paquetes necesarios para el correcto funcionamiento de Flask.

Por último, solo será necesario configurar la nueva web de la que apache tiene que encargarse creando un archivo (en este ejemplo se llamará `renderUI.conf`) en el directorio `/etc/apache2/sites-available`. Dentro de este hay que escribir la siguiente información:

```
<VirtualHost *:80>
  ServerName <nombre de dominio del servidor>
  ServerAdmin admin@<nombre de dominio del servidor>
  WSGIScriptAlias / /var/www/renderUI/flaskapp.wsgi

  <Directory /var/www/renderUI/FlaskApp/>
    Order allow,deny
    Allow from all
  </Directory>
```

```
  Alias /static /var/www/renderUI/FlaskApp/static
  <Directory /var/www/renderUI/FlaskApp/static/>
    Order allow,deny
    Allow from all
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/renderUI-error.log
  LogLevel warn
  CustomLog ${APACHE_LOG_DIR}/renderUI-access.log combined
</VirtualHost>
```

Una vez hecho esto, se reinicia el servidor Apache con el comando indicado anteriormente y la interfaz web debería estar disponible a través del nombre de dominio que se ha configurado. En caso de que no se disponga realmente de ese dominio, se puede añadir una línea al archivo `/etc/hosts` de manera que las peticiones que se hagan a ese dominio se redirijan directamente al servidor deseado. Esta solución solo es útil en entornos totalmente controlados en los que se tiene acceso a todas las máquinas que van a utilizar el servicio.

Como se ha comentado con anterioridad, el servidor web puede estar implementado utilizando cualquier lenguaje de programación o framework. Lo más sencillo es utilizar el script `server-node.py` para enviar los mensajes de órdenes de trabajo al sistema de nodos, para chequear el estado de un trabajo, o para recuperar el resultado del mismo una vez acabado.

Este script se encarga de comunicarse con la base de datos Redis y de mover los archivos necesarios entre el servidor y el nodo maestro, para el correcto funcionamiento del programa. En caso de que el servidor web, la base de datos Redis, o el nodo maestro no se encuentre en la misma máquina, este script será el encargado de crear el túnel SSH necesario para la comunicación con la base de datos de una manera segura, y del uso del comando SCP para mover los ficheros.

El script utiliza el archivo de configuración `server_node.cfg` para obtener determinados parámetros que utiliza en su ejecución. Si el archivo no existe, el script lo crea con unos valores por defecto, y en caso de que el archivo no sea válido, se ignora.

Este archivo se basa en el formato JSON y tiene los siguientes parámetros:

- db_host: Hostname o IP de la máquina en la que se ejecuta la base de datos
- db_user: Usuario del servicio SSH en la máquina en la que funciona la base de datos Redis (por defecto el usuario es `render-node`)
- db_ssh_port: Puerto del servicio SSH de la máquina en la que funciona la base de datos Redis (por defecto se utiliza el puerto 22)
- db_redis_port: Puerto del servicio de Redis (por defecto se utiliza el puerto 6379)
- db_redis_pass: Contraseña de acceso a la base de datos Redis (por defecto es null)
- master_host: Hostname o IP de la máquina en la que funciona el nodo maestro
- master_ssh_port: Puerto del servicio ssh de la máquina del nodo maestro (por defecto se utiliza el puerto 22)
- master_dir: Directorio en el que el nodo maestro deposita los archivos de resultados
- server_dir: Directorio desde el que el servidor web muestra los resultados a los usuarios. Este directorio suele estar dentro de la carpeta `static` del servidor web.

Para el correcto funcionamiento del nodo del servidor y, en el caso de que la base de datos o el nodo maestro estén situados en una máquina distinta al servidor, será necesario disponer de acceso por SSH a las mismas. En la sección A.4 de este mismo apéndice se explica cómo generar y exportar un par de claves pública/privada desde un nodo para automatizar el acceso.

El script tiene un modo de ejecución de "test", que permite comprobar si la configuración actual permite un funcionamiento correcto y las comunicaciones y accesos al resto de nodos del sistema están disponibles. Estas comprobaciones se realizan con el comando: `server-node.py --test`.

A.1.1. Ejemplo de archivo: "server_node.cfg"

En este ejemplo el servidor web y nodo maestro están en la misma máquina, y la base de datos Redis se encuentra en otro servidor accesible a través de la red.

```
{
    'db_host': '12.34.56.78',
    'db_user': 'render-node',
    'db_ssh_port': 22,
    'db_redis_port': 6379,
    'db_redis_pass': 'N0tASecureP@ssword',

    'master_host': 'localhost',
    'master_ssh_port': null,
    'máster_dir': '/home/render-node/finished',

    'server_dir': '/var/www/render/static/finished'
}
```

A.2. Base de datos Redis

La instalación de la base de datos Redis es bastante sencilla, y apenas exige configuración. El script `redis_db.py` se encarga de comprobar que la base de datos está instalada y de hacerla funcionar según los parámetros presentes en el archivo de configuración `redis_db.cfg`. En caso de que el archivo no exista, se crea y se inicializa utilizando unas opciones preconfiguradas de seguridad.

El archivo de configuración se basa en el formato de JSON y tiene las siguiente opciones:

- db_port: Puerto en el que ejecutar el servicio de la base de datos (por defecto se usa el puerto 6379)
- db_pass: La contraseña de conexión a la base de datos. En el caso de no existir, el script genera una contraseña aleatoria de 32 caracteres de longitud, utilizando letras mayúsculas, minúsculas, números y algunos símbolos (salvo que se utilice el argumento –no-pass al ejecutar el script)

En el caso de que se quieran utilizar opciones más avanzadas en la configuración de la base de datos no se podrá utilizar este script y habrá que iniciarla con el siguiente comando:

```
$ redis-server path/to/redis.cfg
```

En la página web de la base de datos Redis (http://redis.io/) hay información muy completa sobre las opciones de configuración de las que dispone la base de datos.

A.2.1. Ejemplo de archivo: “redis_node.cfg”

```
{
    'db_port': 6379,
    'db_pass': 'N0tASecureP@ssword'
}
```

A.3. Nodos del sistema

Los nodos necesitan un directorio de trabajo en el que dispongan de permisos de lectura y escritura, para crear los archivos necesarios durante los trabajos a realizar. No es recomendable situar esta carpeta en un sistema de ficheros temporal, para evitar que un fallo repentino pueda causar pérdidas en el trabajo realizado. El método más sencillo para preparar este entorno de ejecución para un nodo es crear un usuario nuevo en la máquina donde se quiere hacer funcionar, y utilizar una carpeta dentro del `/home` de este usuario como directorio raíz del script del nodo. En nuestros ejemplos se ha utilizado el nombre de usuario `render-node` en todas las máquinas del sistema para simplificar el mantenimiento a futuro.

La diferencia existente entre los dos tipos de nodos del sistema es que los nodos trabajadores necesitan tener acceso al nodo maestro, para poder mover ficheros cuando les sea asignado un trabajo.

Por otro lado, todos los nodos necesitan tener acceso a la máquina donde se ejecute la base de datos, para poder crear un túnel SSH y disponer de conexión al sistema de comunicación. En la sección A.4 de este mismo apéndice se explica cómo generar y exportar un par de claves pública/privada de un nodo para automatizar el acceso por SSH

A.3.1. Nodo maestro

Este nodo solo necesita tener acceso SSH a la máquina en la que funciona la base de datos Redis. Por otro lado, en este nodo se almacenarán los trabajos finalizados hasta que sean requeridos por el nodo servidor, o hasta que se eliminen pasado un tiempo, por lo que es recomendable que esta máquina no tenga problemas de falta de espacio de almacenamiento.

Sus opciones básicas de configuración las carga desde el fichero `master_node.cfg`, y son las siguientes:

- db_host: Hostname o IP de la máquina en la que se ejecuta la base de datos.
- db_user: Usuario del servicio SSH en la máquina en la que funciona la base de datos Redis (por defecto el usuario es `render-node`).
- db_ssh_port: Puerto del servicio SSH de la máquina en la que funciona la base de datos Redis (por defecto se utiliza el puerto 22)
- db_redis_port: Puerto del servicio de Redis (por defecto se utiliza el puerto 6379).
- db_redis_pass: Contraseña de acceso a la base de datos Redis (por defecto es null).
- home_dir: Directorio que el script utilizará como raíz para almacenar los archivos generados por los trabajos.

A.3.1.1. Ejemplo de archivo: “master_node.cfg”

```
{
    'db_host': '12.34.56.78',
    'db_user': 'render-node',
    'db_ssh_port': 22,
    'db_redis_port': 6379,
    'db_redis_pass': 'N0tASecureP@ssword',

    'home_dir': '/home/render-node/finished',
}
```

A.3.2. Nodos trabajadores

Este tipo de nodos es el que más dependencias requiere, puesto que es el nodo que se encarga de realizar la mayoría de los trabajos finales. La configuración básica es similar a la de otros nodos, puesto que necesita tener acceso al nodo de la base de datos para comunicarse, y al nodo maestro para mover los ficheros que necesitará para llevar a cabo sus tareas. Los parámetros de la configuración son los siguientes:

- db_host: Hostname o IP de la máquina en la que se ejecuta la base de datos
- db_user: Usuario del servicio SSH en la máquina en la que funciona la base de datos Redis (por defecto el usuario es `render-node`)
- db_ssh_port: Puerto del servicio SSH de la máquina en la que funciona la base de datos Redis (por defecto se utiliza el puerto 22)
- db_redis_port: Puerto del servicio de Redis (por defecto se utiliza el puerto 6379)
- db_redis_pass: Contraseña de acceso a la base de datos Redis (por defecto es null)
- master_host: Hostname o IP de la máquina en la que funciona el nodo maestro

- master_ssh_port: Puerto del servicio ssh de la máquina del nodo maestro (por defecto se utiliza el puerto 22)
- master_dir: Directorio en el que el nodo maestro deposita los archivos de resultados
- home_dir: Directorio que el script utilizará como raíz para almacenar los archivos generados por los trabajos.
- cycles_bin: Ruta del ejecutable cycles en esta máquina.

Este nodo tiene que tener disponible el ejecutable compilado de "Cycles", así como las dependencias que exija dependiendo del estado del sistema operativo de la máquina.

Para compilar el renderizador en nuestro nodo trabajador necesitamos disponer del código fuente del mismo. Conseguirlo es tan sencillo como ejecutar el siguiente comando:

```
$ git clone git://git.blender.org/cycles.git
```

Con esto tendremos una copia del repositorio en nuestra máquina, y solo hará falta movernos al directorio donde se haya descargado el repositorio y compilar con:

```
$ make
```

Previamente se deben instalar las dependencias necesarias, que son las siguientes:

- OpenGL
- GLEW
- Boost (version superior a 1.48)
- OpenImageIO (mas todas las librerías de las que dependa)

- PugiXML (a menos que “OpenImageIO” esté compilada con la versión incluida de esta librería)

El proceso de instalar las dependencias varía mucho en función de los diferentes sistemas en los que se haga, y los nombres de los paquetes no son los mismos. Es por eso que este proceso no se explica detalladamente, y cada usuario deberá preocuparse de solventar los problemas que puedan aparecer.

A.3.2.1. Ejemplo de archivo: “slave_node.cfg”

```
{
    'db_host': '12.34.56.78',
    'db_user': 'render-node',
    'db_ssh_port': 22,
    'db_redis_port': 6379,
    'db_redis_pass': 'N0tASecureP@ssword',

    'master_host': '65.65.45.65',
    'master_ssh_port': '22,
    'máster_dir': '/home/render-node/finished',

    'home_dir': '/home/render-node/temp'
    'cyces_bin': 'bin/cycles'
}
```

A.4. Generación de pares de claves SSH

Para permitir que los nodos puedan acceder unos a otros automáticamente sin interacción por parte del usuario, es necesario generar pares de claves públicas/privadas y exportarlas entre ellos, de manera que a la hora de identificarse en el acceso no sea necesaria una contraseña, y se utilicen estas claves en su lugar.

A continuación se detallan los pasos necesarios para la creación de un par de claves pública y privada en un nodo, y la exportación a otro de la clave pública. Los pares de claves se pueden generar en cualquier máquina, pero dada su importancia hay que asegurarse de que a la hora de trasmitirlas entre máquinas, esto se hace de forma segura.

1. Crear el par de claves RSA.

   Para generar las claves solo hace falta utilizar el comando:

   ```
   $ ssh-keygen -t rsa
   ```

   Cuando nos pregunte por el archivo en el que se quieren almacenar las claves, lo más recomendable es utilizar el que viene por defecto: `/home/username/.ssh/id_rsa`.

   Hay que asegurarse de que no se introduce ninguna contraseña a la hora de generar la clave. Si se le asigna una contraseña a la clave, esta sería necesaria cada vez que se quisiese utilizar la clave para identificarse, no siendo útil en nuestro caso.

2. Exportar la clave pública a otro nodo del sistema.

   Para esto disponemos de dos opciones, la primera y más sencilla es usar el comando:

   ```
   $ ssh-copy-id username@hostname
   ```

En sistemas en los que no se disponga de este comando se puede hacer utilizando una conexión SSH de la siguiente manera:

```
$ cat ~/.ssh/id_rsa.pub | ssh username@hostname \
"mkdir -p ~/.ssh && cat >>  ~/.ssh/authorized_keys"
```

Una vez exportadas las claves, es posible acceder mediante SSH sin que en el momento del login nos soliciten una contraseña. Esto permite que los programas de los nodos funcionen sin interrupción, conectándose y desconectándose al nodo maestro, o al nodo de la base de datos.

Bibliografía

Apache
https://httpd.apache.org/

Blender
https://www.blender.org/

Bootstrap
http://getbootstrap.com/

Cycles Standalone
http://wiki.blender.org/index.php/Dev:2.6/Source/Render/...
https://developer.blender.org/diffusion/C/

Flask
http://flask.pocoo.org/

Luxrender
http://www.luxrender.net

Matplotlib
http://matplotlib.org/

Mitsuba
http://www.mitsuba-renderer.org/

Numpy
http://www.numpy.org/

Python Docs
https://docs.python.org/

RabbitMQ
https://www.rabbitmq.com/

Redis
http://redis.io/

RenderMan
http://renderman.pixar.com/

SciPy
http://www.scipy.org/

StackOverflow
https://stackoverflow.com/

Twisted
https://twistedmatrix.com/trac/

Vispy
http://vispy.org/

Wikipedia
https://wikipedia.org/

Lista de acrónimos

AJAX ..... *Asynchronous JavaScript And XML*
JavaScript Asíncrono y XML

AMQP ..... *Advanced Message Queuing Protocol*
Protocolo Avanzado de Cola de Mensajes

CPU ..... *Central Processing Unit*
Unidad Central de Procesamiento

CSS ..... *Cascading Style Sheets*
Hojas de Estilos en Cascada

CSV ..... *Comma-Separated Values*
Valores Separados por Comas

EC2 ..... *Elastic Cloud Computing*

GPU ..... *Graphics Processing Unit*
Unidad de Procesamiento Gráfico

IaaS ..... *Infraestructure as a Service*
Infraestructura como Servicio

JSON ..... *JavaScript Object Notation*
Notación de Objetos de Javascript

PaaS ..... *Platform as a Service*
Plataforma como Servicio

RGB ..... *Red Green Blue*
Rojo Verde Azul

S3 ..... *Simple Storage Service*

SaaS ..... *Software as a Service*
Software como Servicio

SCP ..... *Secure Copy*

SSH ..... *Secure Shell*

XML ..... *eXtensible Markup Language*
Lenguaje de Marcas Extensible