

Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros



Arquitectura *Big Data* de colas para el procesamiento de vídeo en tiempo real

Presentado por José Luis Garrido Labrador
en Universidad de Burgos — 9 de julio de 2020

Tutor: Dr. Álgvar Arnaiz González y Dr. José
Francisco Díez Pastor

Universidades de Burgos, León y Valladolid



Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

Dr. D. Álgvar Arnaiz González y Dr. D. José Francisco Díez-Pastor, profesores del departamento de Ingeniería Informática.

Exponen:

Que el alumno D. José Luis Garrido Labrador, con DNI 71707244Y, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros titulado *Arquitectura Big Data* de colas para el procesado de vídeo en tiempo real.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 9 de julio de 2020

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

Dr. D. Álgvar Arnaiz González

Dr. D. José Francisco Díez Pastor

Resumen

La población del primer mundo está paulatinamente más envejecida y las enfermedades degenerativas, como el *Parkinson*, afectan cada vez a una mayor parte de los ancianos. Además, los pacientes de estas enfermedades tienen frecuentemente limitaciones para poder desplazarse hasta los centros de salud donde poder recibir la terapia ocupacional que necesitan para hacer su vida normal.

Ante esta situación se desarrolla el proyecto *Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para prevención de caídas en la enfermedad de Parkinson* en el cual se incorpora este trabajo.

El objetivo de este TFM está en desarrollar un sistema de colas que sea escalable, versátil y fiable, dentro de un entorno Big Data, que dé soporte al análisis en tiempo real de los ejercicios de rehabilitación realizados por los pacientes. Se pretende así facilitar la tarea de los terapeutas ocupacionales, aumentar su alcance y que los ejercicios se puedan realizar de manera autónoma, siendo el sistema el que se encargue de comprobar si se están realizando correctamente.

Descriptores

Big Data, colas, *Apache Kafka*, *Apache Spark*, gestión de flujos, *Docker* vídeo, Enfermedad del *Parkinson*.

Abstract

The population of the first world is getting older and therefore degenerative diseases, such as Parkinson, are affecting more and more frequently to elderly people. In addition, patients with these diseases suffer from limitations in being able to travel to health centres in order to receive the occupational therapy they need to lead a normal life.

In view of this situation, the project *Feasibility and cost-effectiveness study of the use of telemedicine is being developed with a multidisciplinary team for the prevention of falls in Parkinson's disease* in which this work is incorporated.

The aim of this project is to develop a queuing system that is scalable, versatile and reliable, within a Big Data environment, which supports real-time analysis of the rehabilitation exercises performed by patients. This is intended to facilitate the task of occupational therapists, increase their scope and that the exercises can be carried out autonomously with the system being responsible for checking that they are being carried out correctly.

Keywords

Big Data, queues, Apache Kafka, Apache Spark, streaming management, Docker, video, Parkinson's disease.

Índice general

Índice general	III
Índice de figuras	V
Índice de tablas	VII
 Memoria	 1
1. Introducción	3
1.1. Material adjunto	4
2. Objetivos del proyecto	5
2.1. Objetivos generales	5
2.2. Objetivos técnicos	5
2.3. Objetivos personales	6
3. Conceptos teóricos	7
3.1. Computación distribuida	7
3.2. Flujos de datos	8
3.3. Procesamiento de imágenes	9
4. Técnicas y herramientas	13
4.1. Gestión de flujo	13
4.2. Herramientas para el procesado de imágenes	15
4.3. Infraestructura de bajo nivel	19
4.4. Herramientas generales	20

5. Aspectos relevantes del desarrollo del proyecto	21
5.1. Desarrollo de la aplicación web	21
5.2. Implementación del flujo	24
5.3. Proceso de despliegue	28
5.4. Extensibilidad	30
6. Trabajos relacionados	35
6.1. Literatura científica relacionada	35
6.2. Aproximaciones en problemas similares	36
7. Conclusiones y Líneas de trabajo futuras	39
7.1. Conclusiones	39
7.2. Líneas futuras	40
 Apéndices	 41
Apéndice A Plan de Proyecto Software	43
A.1. Introducción	43
A.2. Planificación temporal	43
A.3. Estudio de viabilidad	49
Apéndice B Especificación de diseño	53
B.1. Introducción	53
B.2. Diseño procedimental	53
B.3. Diseño arquitectónico	54
Apéndice C Documentación técnica de programación	57
C.1. Introducción	57
C.2. Estructura de directorios	57
C.3. Manual del programador	58
C.4. Compilación, instalación y ejecución del proyecto	61
C.5. Fallos y soluciones	63
 Bibliografía	 65

Índice de figuras

2.1. Flujo ETL objetivo del proyecto.	6
3.2. Diferentes aproximaciones para procesar un flujo de datos ejemplificado con el tamaño de cinco instancias. A la izquierda se puede observar el uso de ventanas deslizantes donde en cada turno se agrupan las últimas instancias. A la derecha se encuentra la aproximación por <i>chunks</i> en el que se cogen grupos de cinco instancias sin compartirse entre bloques. . . .	8
3.3. Espacio de color HSV y la representación geométrica de cada componente.	10
3.4. Ejemplo visual del cálculo de α y β para una imagen	12
4.5. Mapa de calor de la media de espacio consumido (en kilobytes) por fotograma.	17
4.6. Mapas de calor de los tiempos (en milisegundos) de compresión y descompresión.	18
5.7. Funcionalidades del mando de SNES para el control de la aplicación web por parte del paciente.	22
5.8. Cliente para el paciente sobre el soporte instalado en el domici- lio del paciente.	22
5.9. Menú del terapeuta	23
5.10. Máquinas virtuales <i>Docker</i> que implementan el flujo.	29
5.11. Distribución de los datos temporales para el proceso del flujo sobre una imagen.	31
5.12. Gráficos de bigotes con la distribución estadística de los datos temporales para el proceso del flujo sobre una imagen.	32
5.13. Gráficos de bigotes con la distribución estadística de los datos temporales para el proceso del flujo completo sobre una imagen. . . .	34
6.14. Diagrama de la arquitectura presentada por Amit Baghel. . . .	37

B.1. Diagrama de secuencia para el proceso general de la aplicación.	54
B.2. Implementación del flujo ETL para el procesado de imágenes en tiempo real.	55
B.3. Diagrama de despliegue de las máquinas virtuales <i>docker</i>	56
C.1. Árbol de directorios	58

Índice de tablas

5.1. N.º de <i>workers</i> necesarios según el proceso a realizar.	33
A.1. Tareas del <i>sprint</i> 0	44
A.2. Tareas del <i>sprint</i> 1	45
A.3. Tareas del <i>sprint</i> 2	45
A.4. Tareas del <i>sprint</i> 3	46
A.5. Tareas del <i>sprint</i> 4	46
A.6. Tareas del <i>sprint</i> 5	47
A.7. Tareas del <i>sprint</i> 6	48
A.8. Tareas del <i>sprint</i> 7	48
A.9. Costes de personal.	49
A.10. Costes de <i>hardware</i>	49
A.11. Costes de servicios.	50
A.12. Costes totales.	50

Memoria

Introducción

La creciente facilidad para el acceso a Internet de alta velocidad por parte de la población del primer mundo, ha permitido que los servicios de salud puedan llegar de manera telemática a la población alejada físicamente de los centros de salud. Gracias a ello, algunas tareas, como la rehabilitación de pacientes de la enfermedad de Parkinson, se puede realizar con los modernos sistemas de videoconferencia.

Esta nueva forma de conexión entre terapeutas ocupacionales y sus pacientes puede ser apoyada con técnicas de inteligencia artificial. Estas pueden asistir al terapeuta con su tarea rehabilitadora con información en tiempo real, incluso ayudar a la rehabilitación de muchos más pacientes a la vez.

Para esta tarea, es necesaria tanto una determinada infraestructura técnica capaz de soportar y gestionar correctamente la carga de datos en tiempo real, como unos determinados modelos de inteligencia artificial capaces de procesar la información, todo ello con un coste mínimo en tiempo y memoria.

Con estas características se supervisaría la correcta ejecución de las terapias de manera autónoma evitando que una escasez de terapeutas ocupacionales impidiese ofrecer la rehabilitación a los pacientes.

El objetivo de este trabajo consiste en el diseño y la creación de la arquitectura de colas para el procesamiento de los vídeos de los pacientes en tiempo real en un entorno *Big Data*. Además del aquí presentado existe otro TFM, desarrollado por el alumno José Miguel Ramírez Sanz, cuyo objetivo es la creación del algoritmo para el procesamiento del vídeo.

Este trabajo, junto con el de José Miguel, se encuentran dentro del proyecto *Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para prevención de caídas en la enfermedad de Parkinson* del Ministerio de ciencia, innovación y universidades. Expediente **PI19/00670**.

1.1. Material adjunto

Junto a esta memoria se incluyen

- **Anexos** donde se incluyen:
 - Plan de proyecto
 - Diseño del sistema
 - Manual para el programador
 - Manual para el usuario realizado junto a José Miguel Ramírez Sanz
- *Scripts* para el despliegue del sistema de colas para una instalación sobre máquinas *Docker*.

Además se puede acceder a través de Internet al [repositorio GitHub del proyecto](#).

Objetivos del proyecto

Los objetivos del proyecto se han dividido en tres, siendo estos los objetivos generales, los técnicos y los personales.

2.1. Objetivos generales

- Exploración de las diferentes herramientas existentes para el procesamiento de vídeo en tiempo real a través de las fases de emisión, recogida, encolado, ingestión, procesado, enriquecimiento y almacenamiento.
- Estudio del estado del arte en análisis de imagen para el diagnóstico y tratamiento de enfermedades ante distintos escenarios, tanto en aspectos técnicos (iluminación, enfoque...), como en aspectos físicos (resolución, tasa de refresco...).
- Implementación del preprocesado de los vídeos que garanticen la privacidad de los pacientes y la optimización en los procesados posteriores.
- Implementación del software necesario para la recogida de vídeo en tiempo real sobre sistemas de videoconferencia.

2.2. Objetivos técnicos

- Crear una infraestructura software basada en contenedores *Docker* para ser independientes del software anfitrión y facilitar su despliegue.

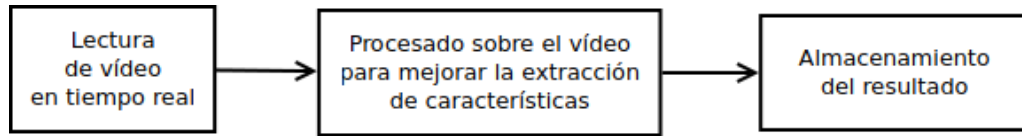


Figura 2.1: Flujo ETL objetivo del proyecto.

- Desplegar un *pipeline* sobre herramientas de la suite de *Apache* para *Big Data* que satisfagan el flujo de extracción, transformación y carga (ETL por sus siglas en inglés) propuesto en la [Figura 2.1](#).
- Conectar el flujo con un algoritmo sobre *Spark Stream* que procese los vídeos, generando los datos necesarios para los estudios posteriores.

2.3. Objetivos personales

- Contribuir a la mejora de la calidad de vida, a través de facilitar soportes para la telerrehabilitación, de pacientes con enfermedad de Parkinson.
- Conocer más profundamente las herramientas de la suite de *Apache* y cómo estas se pueden combinar para permitir el procesado de tareas de *Big Data*.
- Completar mi formación durante el máster a través de la creación de una solución que utiliza gran parte de los conocimientos adquiridos en el mismo.

Conceptos teóricos

En este capítulo se explicarán los conceptos teóricos subyacentes a todos los apartados del proyecto. Se afrontará en tres bloques: primero respecto a la teoría sobre la computación distribuida ([Sección 3.1](#)), otro sobre los flujos de datos ([Sección 3.2](#)) y, por último, sobre la manipulación de imágenes ([Sección 3.3](#)).

3.1. Computación distribuida

Se denomina computación distribuida a las técnicas software para la gestión y el desarrollo sobre sistemas en los que sus componentes físicos y lógicos están separados y se comunican físicamente por red y lógicamente por paso de mensaje [\[35\]](#).

Es importante destacar que la computación distribuida no tiene una relación directa con la computación paralela [\[24\]](#), es decir, aquella que se ejecuta en diversos hilos de ejecución.

La computación paralela no tiene por que ser distribuida ni viceversa. Sin embargo, es habitual que la computación distribuida sea paralela o el servicio se dé paralelamente a diferentes aplicaciones. Un caso típico de computación distribuida y paralela puede ser el de un servicio web. Cada conexión con el servidor es un hilo diferente de ejecución y los recursos necesarios para ofrecer el servicio (bases de datos, procesadores de información, etc.) estén distribuidos en la red.

Debido a que los sistemas que procesan *Big Data* necesitan una gran cantidad de almacenamiento junto con herramientas que lo gestionen de la

manera más eficiente posible, la mayoría de estos sistemas son distribuidos y paralelos.

3.2. Flujos de datos

Un flujo de datos es una estructura de datos continua con elementos ilimitados que aparecen en orden secuencial y no permite acceso aleatorio a los datos. Por este motivo el conjunto de datos completo, ni siquiera una gran parte de los mismos, pueden ser incluidos en la memoria principal. Como los datos son continuos se pueden procesar individualmente, en bloques (también denominados *chunks*) o por ventanas deslizantes. En la [Figura 3.2](#) se puede observar el funcionamiento de las diferentes aproximaciones en el procesamiento del flujo.

Turno	Dato		Turno	Dato	
1	8563		1	8563	
2	6277		2	6277	
3	8852	Ventana 1	3	8852	Bloque 1
4	7975	Ventana 2	4	7975	
5	6550	Ventana 3	5	6550	
6	4143	Ventana 4	6	4143	
7	4039	Ventana 5	7	4039	Bloque 2
8	4559	Ventana 6	8	4559	
9	5084		9	5084	
10	3747		10	3747	

Figura 3.2: Diferentes aproximaciones para procesar un flujo de datos ejemplificado con el tamaño de cinco instancias. A la izquierda se puede observar el uso de ventanas deslizantes donde en cada turno se agrupan las últimas instancias. A la derecha se encuentra la aproximación por *chunks* en el que se cogen grupos de cinco instancias sin compartirse entre bloques.

Por este motivo los algoritmos que procesan flujos de datos han de tener dos características principales [11]:

1. deben procesar cada parte del flujo que reciban antes de que llegue la siguiente
2. y deben procesar los datos usando una pequeña cantidad de memoria.

Debido a esto, los algoritmos que se ejecutan sobre los flujos se suelen enfocar a soluciones aproximadas para minimizar el uso temporal y espacial. Además, si se aplican algoritmos cuyo comportamiento dependa de la evolución del flujo y el contenido del mismo cambia en sus características estadísticas, el programa debe poder adaptarse a estos cambios.

Para la adaptación de cambios existen dos aproximaciones principales: el uso de ventanas deslizantes, para que de este modo solo se tengan en cuenta los datos más recientes, y el uso de ventanas de decaimiento, en el que se da más peso a los datos más recientes.

Aplicaciones

Los flujos de datos aparecen en multitud de entornos, principalmente en aquellos donde el tiempo real es más importante y los datos se generan continuamente.

Ejemplos de estas aplicaciones pueden ser [27]:

- Sensores en el *Internet de las cosas* (IoT),
- telecomunicaciones: llamadas, ubicación de los dispositivos . . . ,
- redes sociales: interacciones de los usuarios, tendencias . . . ,
- comercio electrónico: publicidad en tiempo real, actividad del usuario, detección de fraude . . . ,
- asistencia sanitaria: telemedicina, evolución de los pacientes . . . ,
- o epidemias y desastres naturales: evolución de una enfermedad, propagación vírica . . .

3.3. Procesamiento de imágenes

A la hora de hacer procesamiento sobre vídeo, el fundamento principal es el procesamiento de los fotogramas que lo componen. Por tanto es importante entender cómo funcionan a nivel lógico las imágenes digitales, cómo operar con ellas y cómo procesarlas para obtener los mejores resultados.

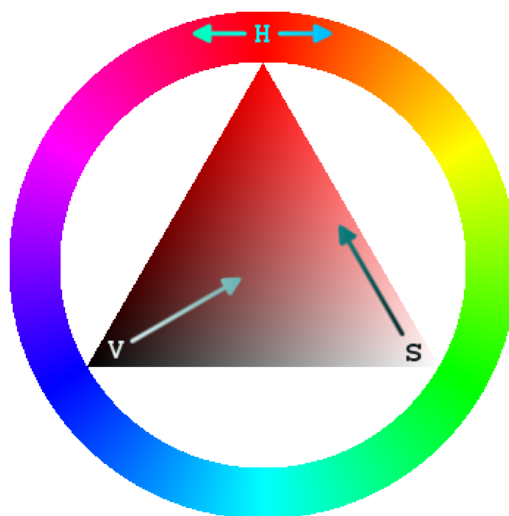


Figura 3.3: Espacio de color HSV y la representación geométrica de cada componente. Fuente: [De Samus_ - Trabajo propio, CC BY-SA 3.0](#)

Definiciones

Imagen de 24 bits: codificación habitual de las imágenes a color. Cada píxel se define como la combinación de tres enteros sin signo de 8 bits. La codificación (o espacio de color) necesaria para la visualización en monitores es la combinación de las capas de color roja, verde y azul, conocida como RGB, aunque también la BGR usada por openCV.

HSV [32]: acrónimo del inglés de matiz, saturación y valor ([Figura 3.3](#)). Consiste en un modelo de color basado en los componentes del mismo y representa una transformación no lineal del espacio de color RGB. El matiz, también conocido como tono, es el grado del ángulo respecto a la rueda de color. Representa un color único y algunos ejemplos son el rojo (0°), amarillo (60°) o verde (120°). La saturación representa la pureza del color entre el gris (valor mínimo) al color puro (valor máximo). El valor representa la luminosidad del color siendo el valor mínimo el color negro y el valor máximo el color con la saturación aplicada.

Histograma [14]: El histograma es una representación de cuántos píxeles hay para cada valor de intensidad. En él se puede ver información muy útil de la imagen como su brillo (media del histograma) o su contraste (diferencia entre el valor significativo¹ máximo y mínimo). Los procesos

¹El valor mínimo y máximo tienden a ser siempre los mismos independientes del histograma (0 y 255 respectivamente), por eso se consideran los valores que tengan una presencia significativa, por ejemplo que sea al menos un 1 % de los valores.

de ajuste del brillo y contraste se pueden observar como transformaciones del histograma.

Ajuste automático de brillo y contraste

Para ajustar el brillo y contraste de una imagen I hay que seleccionar dos valores α y β que se deben aplicar sobre cada píxel de la imagen tal que la ecuación sea:

$$I'(x, y) = \alpha * I(x, y) + \beta \quad (3.1)$$

Para hacer el proceso de manera automática es necesario encontrar los valores de α y β adecuados. El procedimiento consiste en «estirar» una parte del histograma de la imagen de tal manera que el contenido seleccionado del mismo abarque todo el histograma, a esto se llama *corte de alas* y va controlado por un parámetro que decide la posición del corte (Figura 3.4).

Por tanto, siendo p el porcentaje mínimo de la frecuencia del *valor* y \vec{h} el histograma de la capa *valor* de HSV, el proceso es el siguiente [26]:

1. Se calcula la frecuencia acumulada de \vec{h} , lo definimos como el vector \vec{a} .
2. Se actualiza el valor de p con $p = p * \frac{\max(\vec{a})}{100*2}$
3. Se define un valor g_{min} como el valor más alto de \vec{a} que sea menor que p , también se define un valor g_{max} como el valor más pequeño de \vec{a} que sea mayor que $\max(\vec{a}) - p$
4. Se calcula el valor de α como $\alpha = \frac{255}{g_{max} - g_{min}}$. (255 es el valor máximo permitido en la codificación 8 bits.)
5. Se calcula el valor de β como $\beta = -g_{min} * \alpha$.

En el caso de que se quisiera dejar el mismo brillo y solo cambiar el contraste, el valor de α a aplicar en la ecuación (3.1) será 1, en caso opuesto el valor de β deberá ser 0.

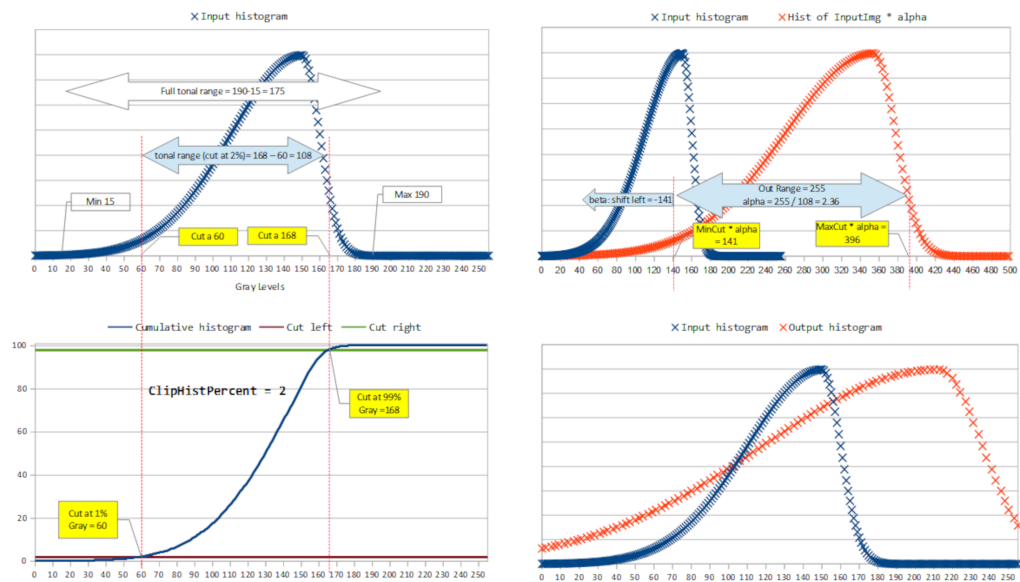


Figura 3.4: Ejemplo visual del cálculo de α y β para una imagen. Fuente: De PKLab - en opencv.org

Técnicas y herramientas

En este capítulo se detallan las técnicas y herramientas utilizadas para la **gestión del flujo** ([Sección 4.1](#)), el **soporte de bajo nivel** ([Sección 4.3](#)) sobre las que se ejecutan los servicios, y las herramientas generales ([Sección 4.4](#)).

Como este trabajo es parte de un proyecto mayor, en la [Sección 5.1](#) se comentan las herramientas para los otros componentes del sistema. En este capítulo se explicarán únicamente las herramientas usadas en este TFM.

4.1. Gestión de flujo

Uno de los puntos esenciales de este trabajo es recoger y dirigir los *streams* de vídeo que se reciben. Por tanto, escoger una correcta aplicación para la gestión de este flujo de datos es una parte muy importante.

Dentro de la suite de *Apache* existen varios componentes que se encargan de la gestión del flujos de datos. Se necesitan dos herramientas principales, una capaz de dirigir el flujo y otra para procesarlo.

Herramientas para dirigir el flujo

La primera herramienta necesaria debe ser capaz de dirigir flujos de datos, concretamente flujos de datos serializados para soportar tipos de datos más diversos como son las imágenes. También se necesita que sea capaz de desplegar diferentes colas para discriminar fácilmente a los distintos flujos de vídeo entrante.

En la suite de *Apache* existen dos herramientas que gestionan flujos de datos, son: *Apache Flume* [1] y *Apache Kafka* [3].

Apache Flume es una herramienta de gestión de flujo diseñada para hacer una gestión distribuida de manera fiable y altamente disponible de los datos. Proporciona un servicio eficiente para la recogida, agregación y almacenamiento de los datos. Sin embargo, aunque esta herramienta pudiese suplir las necesidades de una gestión de flujo, se ha descartado debido a que está optimizada para la gestión de *logs*, concretamente datos codificados como cadenas, y no tiene un sistema de colas.

Apache Kafka es un proyecto de intermediación de mensajes que trabaja sobre el patrón publicación-suscripción, funcionando como un sistema de transacciones distribuidas. Incorpora, para la implementación de este patrón, un sistema de colas para la distribución de mensajes. Aporta una API para el productor, para el consumidor, para el flujo y para el conector. La conexión con *Kafka* se realiza a través del protocolo de la capa de transporte *TCP*. Las diferentes colas que provee *Kafka* se denominan *topics* y son el identificador que utilizan las *APIs* de productor y consumidor para seleccionar la cola sobre la que quieren operar. Para que *Kafka* pueda funcionar, necesita estar conectado a un servidor de *Apache Zookeeper*, encargado de la gestión de la memoria y el despliegue de *Kafka*.

Se elige *Apache Kafka* como herramienta para la dirección del flujo, al aportar un sistema de colas distribuidas fiable y soportar datos serializados.

Herramientas para procesar el flujo

En segundo lugar se necesita una herramienta capaz de procesar flujos de datos de forma escalable. La suite de *Apache* tiene dos herramientas principales para el procesado de información, son: *Apache Hadoop* y *Apache Spark*, ambas con extensiones para el procesado de flujos.

Apache Spark Streaming [4] incluye nativamente una *API* de consumidor de *Kafka* y *Flume*, e integración con los sistemas de ficheros *HDFS* y *S3* entre otros. El funcionamiento interno consiste en crear pequeños lotes de datos para pasarlo al motor de *Spark* y retornar los lotes procesados.

Apache Hadoop Streaming [2] tiene un funcionamiento similar a *Spark Streaming* pero sin aportar de manera nativa una integración con *Kafka* y *Flume*.

Se ha escogido *Spark Streaming* frente a *Hadoop Streaming* por varios motivos:

1. *Spark Streaming* tiene integración con *Kafka* de manera nativa.
2. *Spark Streaming* hace un uso intensivo de la memoria RAM, por lo que es mucho más rápido si se cuenta con un equipo con la suficiente memoria disponible para el trabajo deseado².

Servicio de *streaming* de vídeo

Para comunicar al paciente con el terapeuta asociado y con el sistema de procesamiento de vídeo, se ha utilizado la herramienta *Jitsi* [5]. Consiste en un servicio de código abierto (Apache 2) para construir y desplegar de manera sencilla una solución de conferencias de vídeo.

Se compone de dos partes: *meet* y *videobridge*. *Jitsi-Meet* es la interfaz web de la aplicación y es usada para que el paciente y el terapeuta se comuniquen. *Jitsi-Videobridge* es la infraestructura de recogida y emisión de vídeo y audio que conectan a ambos clientes.

La razón para usar esta herramienta y no cualquier otro servicio de videoconferencia, es porque *Jitsi* es software de código abierto. Gracias a esto podemos desplegar en los servidores locales la herramienta y poder duplicar el vídeo del paciente y que sea dirigido al ingestor de *Kafka*.

4.2. Herramientas para el procesamiento de imágenes

Para el procesamiento de las imágenes que componen los vídeos que se van a analizar existe *OpenCV* [13], una biblioteca de código abierto desarrollada sobre C++ e integrada en otros muchos lenguajes como *Python*. Debido a la libertad que ofrece al ser *open source*, ser muy versátil, tener muchos años de trayectoria, con la consecuente mejora continua de la herramienta, y tener una documentación muy completa, se ha optado directamente por esta herramienta.

Para poder garantizar la privacidad de los pacientes, se ha realizado detección de rostros y su posterior anonimización. Para ello se utiliza un modelo de *Caffe*, biblioteca de visión por computador con una gran variedad de modelos de clasificación y detección [18]. Se usa concretamente el modelo `res10_300x300_ssd_iter_140000`. Consiste en una red neuronal

²El programa final se ha desplegado sobre una máquina con 128 GB de RAM y se ha probado en una de 32 GB de RAM con buenos resultados.

convolucional, principal tipo de red neuronal para el análisis de matrices 2D [21] como son las imágenes.

Elección del sistema de compresión de fotogramas

La principal limitación dentro procesamiento de flujos es mantener un uso bajo de memoria dentro de un periodo de tiempo pequeño que garantice poder atender a cada componente del flujo.

Una de las técnicas para garantizar un menor uso de memoria es la compresión de los *bytes* que se ingestan en el sistema de colas, aunque esto tiene una mayor carga temporal. Además de la compresión para poder introducir datos en un flujo y luego recuperarlos hace falta serializarlos, es decir, convertirlo en una serie de bytes.

Existen multitud de técnicas de serialización y de compresión de secuencias de *bytes*. Para poder decidir qué combinación de estas dos técnicas se van a utilizar, se ha buscado que minimicen el tamaño en memoria y realicen la ejecución en la menor cantidad de tiempo posible.

Para ello se han explorado las siguientes técnicas de serialización:

- Serialización nativa: que usa la herramienta *pickle* propia de *Python* para la serialización de objetos.
- Codificación *JPG/Exif* [25]: estándar de codificación de imágenes que incluye sistema de compresión regulable con pérdidas.
- Codificación *PNG* [12]: estándar de codificación de imágenes con sistema de compresión sin pérdidas.

Debido a que los estándares *JPG/Exif* y *PNG* son regulables, según el parámetro de compresión interno, se han explorado en el caso de *JPG/Exif* la calidad 95 % y 80 % mientras que con *PNG* las calidades 9 y 5.

Junto con estos sistemas de serialización se han combinado los siguientes algoritmos de compresión:

- *LZMA* [10, 31]: evolución del algoritmo *LZ77* para la compresión sin pérdida de flujos de datos, evolucionó con la incorporación de las cadenas de Márkov.

- *Gzip* [6]: implementación libre del algoritmo de compresión *DEFLATED*, extensión de la versión original de *LZMA* (sin cadenas de Márkov).
- *Zlib* [28]: abstracción del algoritmo de compresión *DEFLATED* que usa una menor cantidad de recursos del sistema.

Para hacer la prueba se utilizaron dos vídeos con una duración total entre los dos de 13 minutos, con una frecuencia de 15 fotogramas por segundos, un total de 11 700 fotogramas³. De cada uno de estas imágenes se calculó el tiempo de compresión, el tiempo de descompresión y el tamaño en memoria que ocupaba tras la compresión. De estos resultados se obtuvo el valor medio. En las figuras 4.5 y 4.6 se pueden observar tres mapas de calor con los resultados de cada experimento.

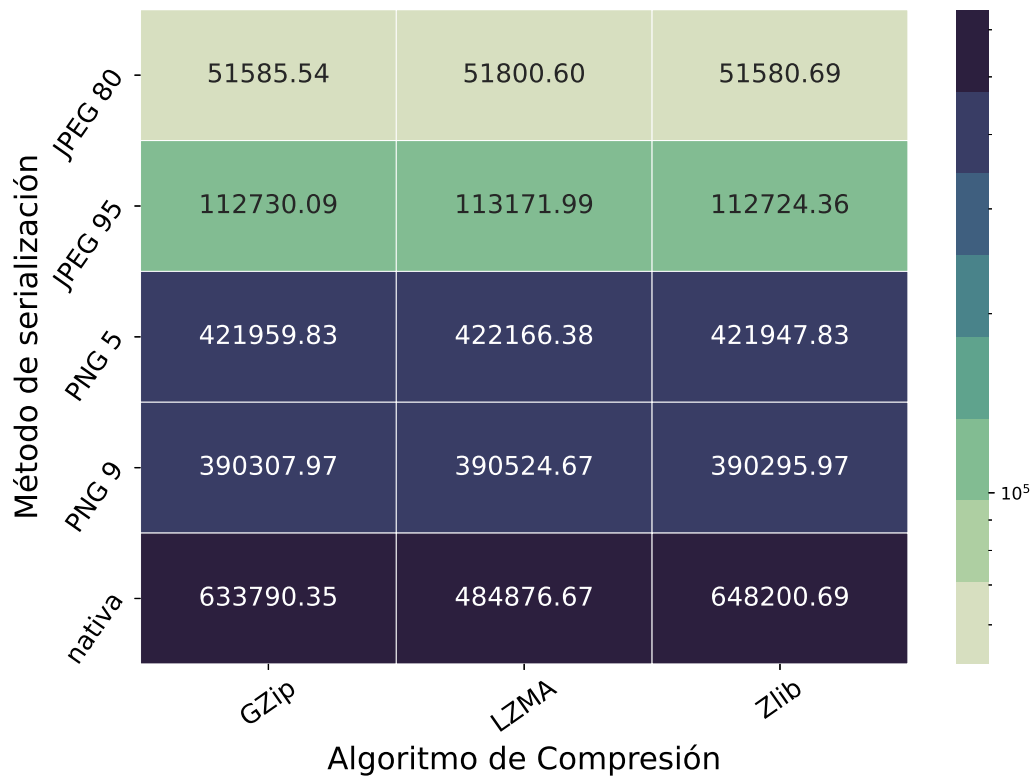
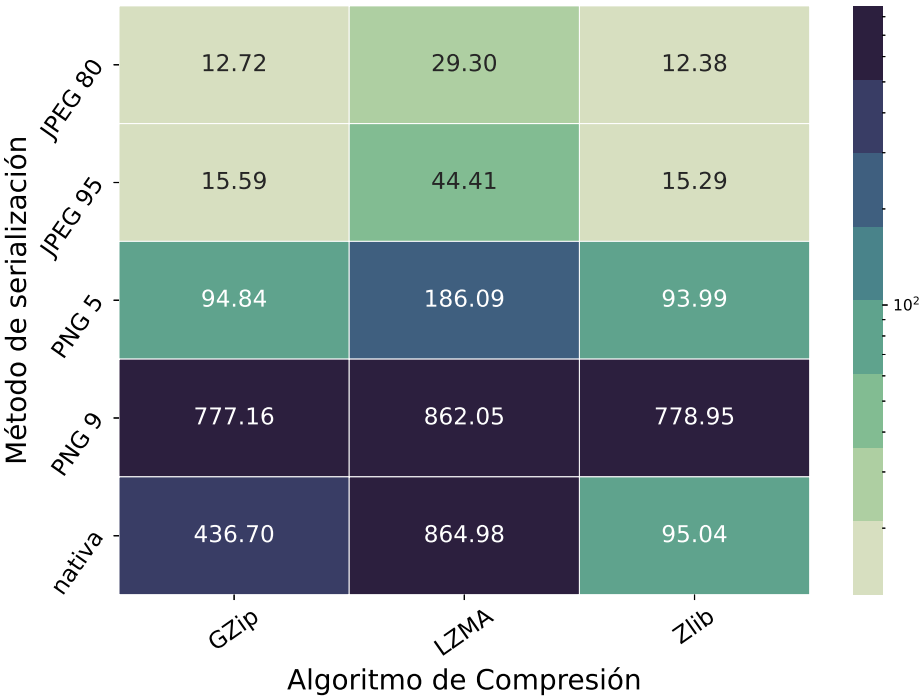
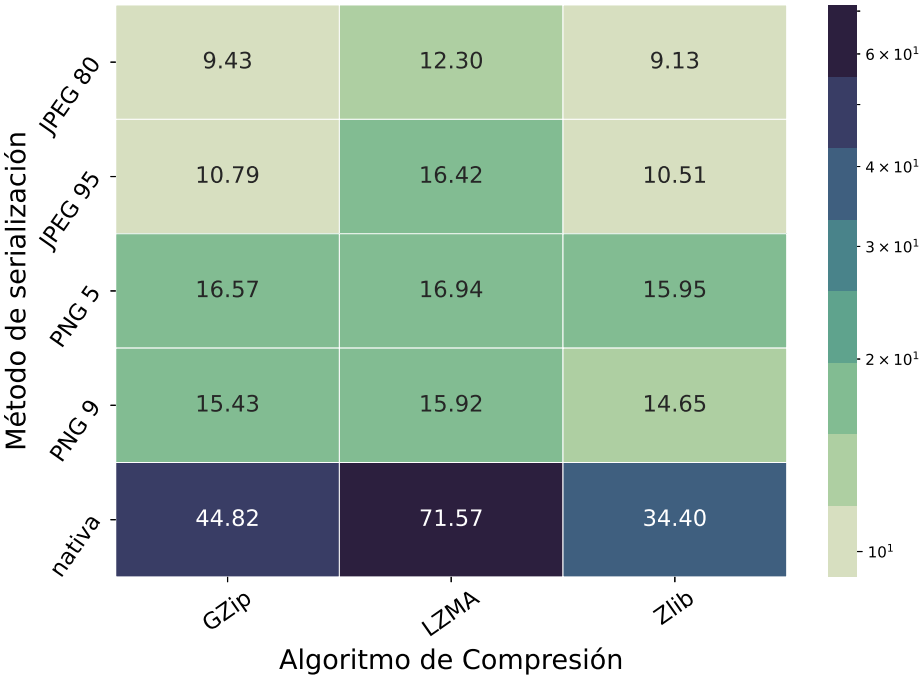


Figura 4.5: Mapa de calor de la media de espacio consumido (en kilobytes) por fotograma.

³De una batería de vídeos grabados con las cámaras de los pacientes.



(a) Media del tiempo necesitado para comprimir fotogramas.



(b) Media del tiempo necesitado para descomprimir fotogramas.

Figura 4.6: Mapas de calor de los tiempos (en milisegundos) de compresión y descompresión.

Se puede observar que el algoritmo de compresión *LZMA*, aunque consigue en la mayoría de las veces una muy alta compresión, requiere siempre el mayor tiempo tanto de compresión como de descompresión. De manera similar, la serialización nativa mantiene siempre un uso mucho mayor de memoria que el resto de serializaciones, debido principalmente porque tanto *JPG* como *PNG* incorporan un compresor adicional.

Respecto a los algoritmos de compresión *GZip* y *ZLib*, las diferencias son muy sutiles tanto en memoria como en tiempo, principalmente porque están basados en el mismo algoritmo.

El uso de *JPG* sobre *PNG* garantiza un uso significativamente menor de memoria aunque a costa de una pérdida de información, sin embargo la pérdida en el caso de *JPEG 95* es despreciable teniendo en cuenta la ventaja que tiene en espacio.

Para este proyecto se utilizarán *JPEG 95* debido a que garantiza una mejor calidad que su contraparte de *JPEG 80* y con una cantidad mucho menor que el resto de serializaciones. Junto a este método se utilizará el algoritmo *ZLib* ya que tiene las mejores métricas temporales.

4.3. Infraestructura de bajo nivel

Otro apartado importante en el despliegue de la aplicación son las herramientas y técnicas a ser usadas para la producción. Para esto se utilizan:

- ***GNU/Linux***, el sistema operativo más extendido en el entorno de los servidores [23, 36] además de estar disponible en los servidores prestados para la realización de este proyecto⁴.
- ***Docker***, un software de gestión de contenedores estandarizados, semejante a los entornos *chroot* que facilita la virtualización de software en un entorno seguro y ligero. Sobre este motor se ejecutarán las aplicaciones del entorno de *Apache Spark* [20], *Apache Kafka* [19] además de la aplicación desarrollada para el cumplimiento de los objetivos.

⁴Se utilizan el servidor *Alpha* del GIR [ADMIRABLE](#) para el despliegue de la herramienta de recolección de datos (Procesador *Intel Core i7-8700*, 6 núcleos, 3.2GHz. 64GB de memoria RAM. 2 GPUs *GTX 1080Ti* y 500GB de disco duro sólido y 6 TB de disco duro magnético) y el servidor *Gamma* del mismo grupo para el despliegue de las colas y el procesamiento de la información (Procesador *Intel Xeon*, 10 núcleos. 128 GB de memoria RAM. 3 GPU *Nvidia Titan Xp*).

- *Ngrok*, software que ofrece un tunel TCP y HTTP sobre un servicio que no tiene acceso al exterior. Es usado para poder acceder al servicio web desde fuera de la universidad.

4.4. Herramientas generales

Para el desarrollo general del proyecto se han utilizado las siguientes herramientas:

- **Visual Studio Code**: Editor de código genérico de código abierto (MIT).
- **Overleaf**: Editor de \LaTeX online para el trabajo colaborativo.
- **TeXStudio**: Editor de \LaTeX de código abierto (GPLv2).
- **Dia**: Editor de diagramas genérico de código abierto (GPL).
- **Filezilla**: Aplicación para la transferencia de ficheros sobre *FTP* y *SFTP* de código abierto (GPLv2).
- **GitHub**: Servicio online de *hosting* para repositorios Git.

Aspectos relevantes del desarrollo del proyecto

En este capítulo se explicarán las partes más importantes del desarrollo del proyecto.

5.1. Desarrollo de la aplicación web

Para realizar la primera fase del desarrollo, la recogida de los datos, se ha desarrollado una aplicación web que facilitase las tareas de tele-rehabilitación conectando al personal especializado con los pacientes con enfermedad del *Parkinson*. Esta recogida se hizo incorporando dentro de sus funciones la grabación a los mismos durante la realización de los ejercicios para ser estudiados en tiempo real.

Dentro del funcionamiento natural del flujo, se incorpora una función de anonimización del rostro para mantener la intimidad de los pacientes en los estudios posteriores sobre el material grabado, además no se guardan datos identificativos de ningún paciente.

Esta aplicación se ha compuesto de dos partes. Una consistió en el diseño y consecuente implementación de una interfaz hombre-máquina fácil de usar y accesible mediante un mando sencillo⁵. Esta se compone de una serie de botones de colores, directamente relacionados con los del mando (figura 5.7), que permiten iniciar una comunicación mediante videoconferencia con el personal terapéutico y finalizar la reunión. Durante

⁵Concretamente un mando de la consola SNES adaptado para ser usado mediante USB.

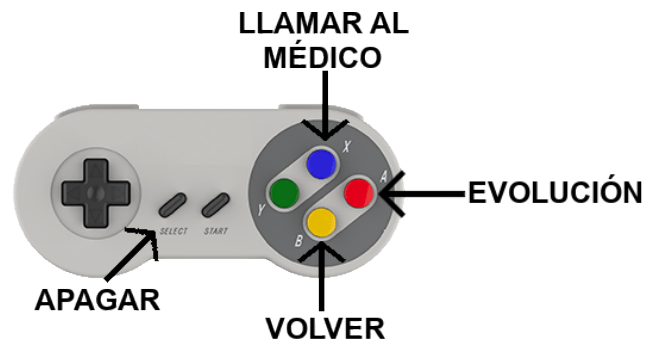


Figura 5.7: Funcionalidades del mando de SNES para el control de la aplicación web por parte del paciente.

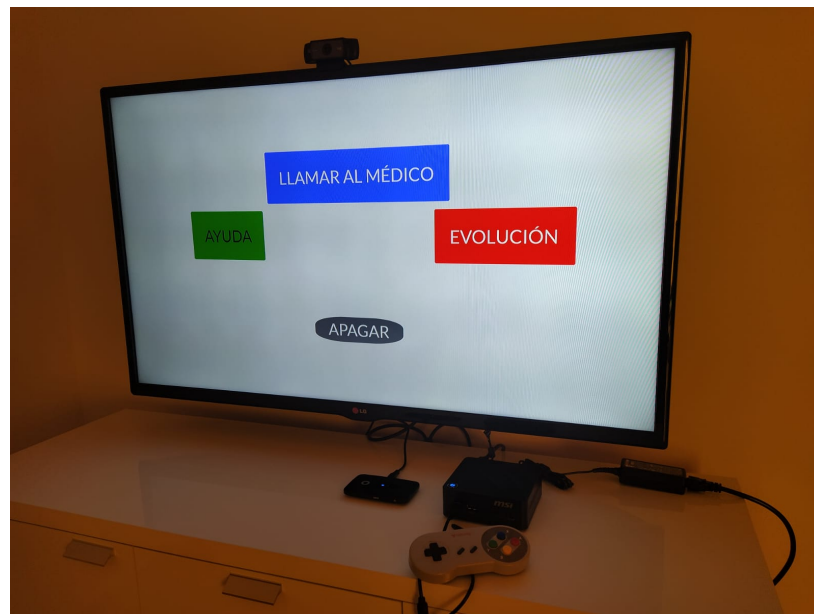


Figura 5.8: Cliente para el paciente sobre el soporte instalado en el domicilio del paciente.

el proceso de la llamada, el vídeo del paciente es capturado y enviado al servidor para su procesado en tiempo real.

En la [Figura 5.8](#) se puede observar el equipo destinado para el uso por parte del paciente ya instalado y listo para usar.

La segunda parte consiste en la interfaz del terapeuta encargado de dirigir las sesiones de rehabilitación del paciente ofreciendo una interfaz sencilla que permite conectarse a las videoconferencias de los diferentes



Figura 5.9: Menú del terapeuta

pacientes y dirigir las sesiones de terapia, además de gestionar la evolución del paciente (figura 5.9).

Las características del equipo donde está desplegada la aplicación del paciente es:

- MSI – Cubi N 8GL-001BEU N4000 1.10GHz.
- WB Green M.2 120 GB SATA 3.
- 4 GB DDR4 2400 MHz.
- Lubuntu x86_64 18.04.
- WebCam Logitech HD Pro C920.

Herramientas utilizadas

Este servicio se divide en tres partes: el *backend*, el *frontend* y el servicio de *streaming* de vídeo.

Backend

El *backend* es el conjunto de servicios en el lado del servidor de un servicio web. Para construirlo se han utilizado dos librerías de *Python*.

Jinja [29]: gestor de plantillas de código abierto (BSD), se utiliza para la creación de la interfaz web mediante la creación de páginas dinámicas en HTML.

Flask [30]: *microframework* de código abierto (BSD) que ofrece una capa de abstracción muy alta de un servicio web, se utiliza para la creación de la lógica de negocio mediante la gestión de las rutas.

FrontEnd

El *frontend* es la parte del servicio web que muestra la información al cliente y permite la interacción con el mismo. Se han usado dos *frameworks* para HTML5.

Bootstrap [33]: *Framework* de CSS de código abierto (MIT) creado por Twitter para la creación de aplicaciones web redimensionables. Todos los estilos de la web se apoyan en este *framework*.

jQuery [34]: *Framework* de JavaScript de código abierto (MIT) que simplifica el acceso al *HTML DOM* de la página. La gestión de los eventos de la web se gestionan mediante este *framework*.

5.2. Implementación del flujo

Los componentes del flujo ETL implementado (Figura B.2) se realizaron mediante tres *scripts* de *Python* según las tres etapas del flujo. Primero un emisor *UDP* de vídeos que simula la entrada de los vídeos de los pacientes. El segundo *script* es el ingestor en *Kafka* de los fotogramas del vídeo recibido. Por último, el tercero es el procesador de los diferentes *frames* mediante *Spark* y *OpenCV*.

El *script* emisor consiste en un cliente *UDP* que itera sobre los fotogramas del vídeo y los comprime mediante la abstracción de *DEFLATE zlib* [28] sobre la imagen serializada bajo el estándar *JPEG/Exif* [25] con una calidad del 95 %. Esta compresión del fotograma permite que la velocidad de encolado se minimice. Además este *script* simula la tasa de refresco de un vídeo real emitiendo a quince fotogramas por segundo, las frecuencia de los vídeos reales que se capturarán con los pacientes⁶.

El segundo *script*, el ingestor, se divide en dos partes, una encargada de la escucha de los paquetes *UDP* que se reciban por parte del emisor, y otra destinada al encolado de los fotogramas en un productor de *Kafka*. El servidor *UDP* se implementa mediante un *socket* de flujo (*SOCK_STREAM*) de tal manera que no es necesario conocer el tamaño del paquete con anterioridad dando así más flexibilidad al flujo completo. Respecto a la

⁶Esto no será necesario una vez se integre completamente con *Jitsi*.

emisión al productor de *Kafka* el fotograma comprimido se envía junto a una clave que indica el número dentro de la secuencia del vídeo.

El último componente es el procesador de las imágenes. Consiste en un flujo de *Spark Streaming* que se conecta a un *topic* de *Kafka*. Este es el punto más extensible del flujo ya que la mayor dificultad está en la conexión previa que se ha de hacer. La configuración básica está en procesar cada *frame* por separado mediante el sistema de *microbatching* que utiliza *Spark*, pero se puede configurar para hacer ventanas deslizantes con los que procesar varios fotogramas. Se incorpora un proceso sobre las imágenes para la anonimización mediante la detección del rostro usando el modelo de *Caffe* [18] y aplicándole un filtro gaussiano o una reducción de resolución sobre la zona detectada (para más información ver la sección C.3). Adicionalmente se ha incorporado un algoritmo de corrección del contraste y del brillo.

Debido al funcionamiento de *Spark Streaming* se pueden acoplar más funciones. Aunque cuantas más sean, más se realentizará el proceso y con ello se puede perder el factor de «tiempo real» en el análisis del flujo. El objetivo con esto es que el trabajo del compañero José Miguel Ramírez Sanz se pueda incluir como el último paso en el procesado del vídeo.

Limitaciones

Debido a la naturaleza de las herramientas utilizadas existen algunas limitaciones importantes a mencionar.

La principal limitación está en el tiempo sobre las acciones alrededor de *Apache Kafka*, ya que existen tres momentos temporales que poseen máximos técnicos. El primero es el tiempo de encolado, si un paquete tarda mucho en ser insertado al *topic* correspondiente este es rechazado, la velocidad de este proceso está fuertemente ligada al tamaño del paquete emitido. El segundo tiempo es el tiempo de vida del paquete dentro de la cola y si un paquete está más tiempo en la cola que este tiempo es automáticamente rechazado. Por último está el tiempo de desencolado, nuevamente un tiempo ligado al tamaño del paquete, sin embargo, aunque un tiempo excesivo no elimina al paquete, sí que ralentizará los pasos posteriores, además este tiempo es el único que no se puede parametrizar en la configuración de *Kafka*.

Otra de las limitaciones es la memoria, tanto memoria *RAM* como *VRAM*⁷. Cada uno de los paquetes emitidos se deben almacenar en *RAM*

⁷Memoria *RAM* para vídeo.

hasta su procesamiento final por lo que en caso de tener una cantidad disponible menor disponible de la necesaria el flujo fallará y los paquetes afectados serán perdidos.

Por último está la limitación temporal en las operaciones de *Apache Spark Streaming*. Una vez los *frames* son consumidos por *Spark* desde la cola de *Kafka* estos paquetes permanecerán en la memoria mientras se procesan, sin embargo, si el proceso consume más tiempo del disponible (considerando tiempo disponible como el intervalo entre dos *frames* en la secuencia original del vídeo) entonces se acumulan las *imágenes* a procesar. Se consume por tanto una mayor cantidad de memoria y se pierde el tiempo real.

Soluciones frente a las limitaciones

Para solucionar estas limitaciones se plantearon diferentes estrategias, algunas con muy buenos resultados y otras fueron descartadas. Estas estrategias fueron las siguientes:

- **Reducir tiempo de encolado:** para garantizar un encolado rápido de los fotogramas se plantearon dos soluciones:
 - *compresión del fotograma* utilizando el algoritmo previamente mencionado *zlib* [28]
 - y *rescalado del fotograma* reduciendo un porcentaje su tamaño manteniendo la proporción de aspecto.

Combinando ambas aproximaciones se garantizó que siempre se encolasen los *frames*. Respecto al tamaño, se utilizó como altura de cada fotograma 480 *píxeles*, ya que así funcionaba de la manera deseada el modelo de inteligencia artificial que se iba a integrar al final. También se comprobó que si el fotograma ocupaba un máximo de 800 *kilobytes* se encolaba siempre sin problemas, de tener más no había garantías.

- **No superar el tiempo de vida en *Kafka*:** para evitar pérdidas de fotogramas se siguieron dos estrategias:
 - *aumentar el tiempo de vida* parametrizando en *Kafka* un tiempo de vida mayor del necesario
 - y *garantizar un consumo en el tiempo de vida* obligando a *Spark* a consumir un *fotograma* antes de que se perdiese.

Ambas soluciones planteadas tienen el mismo efecto en memoria. Sin embargo, se optó por la segunda ya que el tener el fotograma cargado lo antes posible en *Spark Streaming* permitía un proceso más rápido si se aumentaban los recursos de este. Como el proceso de aumentar recursos es tan simple como crear un nuevo *worker*, es preferible esta aproximación que alargar el tiempo de vida en *Kafka*. Además, para garantizar el comportamiento sincronizado, el flujo consumirá un fotograma según la tasa de imágenes por segundo (comúnmente llamado *fps* por sus siglas en inglés).

- **Reducir tiempo de desencolado:** debido a que el problema que se intenta solucionar tiene las mismas causas que el problema de reducir el tiempo de encolado, al aplicar una compresión del fotograma y un rescaldado del mismo se reduce este tiempo.
- **Evitar saturación de la memoria:** este problema depende mucho de los procesos realizados por parte de *Spark*, por lo que realmente solo existe una aproximación y es tener tanta memoria RAM y VRAM como sea necesaria. Sin embargo, siempre se puede reducir el uso de memoria sacrificando algunos *frames*, aunque esta opción no es la más deseable.
- **Garantizar el tiempo real:** para evitar el uso excesivo del tiempo de procesado se plantean dos soluciones:
 - *reducir operaciones de entrada y salida innecesaria* como son las salidas por pantalla, logs o escritura en disco
 - *y aumentar el número de workers de Spark* para distribuir la carga evitando saturaciones.

Respecto a la primera aproximación es evidente la necesidad de reducir el uso en actividades secundarias, ya que la propia teoría del procesamiento sobre flujos de datos especifica las limitaciones de complejidad. Sin embargo, en muchas ocasiones las necesidades de escribir y leer sobre el disco son indispensables, por tanto, lo ideal para este flujo es evitar el almacenamiento de las imágenes procesadas y solo guardar la información relevante que se desee.

Para la segunda solución planteada se requiere una mayor cantidad de recursos *hardware* por lo que existe una limitación de extensión, sin embargo, es esencial contar con los recursos *hardware* necesarios ante procesados de información en entornos *Big Data*. En este proyecto se han combinado ambas soluciones.

Problemas de la documentación oficial

Algo a tener en cuenta en el proceso de despliegue han sido las carencias que ha tenido la documentación oficial, ya que la [guía de integración oficial](#) no cuenta con los detalles para *Python* junto a *Kafka* y otras fuentes que enfrentan problemas similares [9] trabajaban con versiones diferentes de las herramientas usadas.

El principal reto en la implementación del flujo ha sido el obtener los conocimientos suficientes, tras una exploración minuciosa de las herramientas, para poder hacer un correcto desarrollo y alcanzar los objetivos planteados.

5.3. Proceso de despliegue

Para el despliegue del flujo se crearon varias imágenes *docker* junto a una serie de *scripts* en *bash* que las gestionen.

Se utilizan un total de seis contenedores, algunas de ellos con varias instancias, para el total del flujo. Esta son:

- *confluent/cp-zookeeper* [19] encargada de crear un servidor *Apache ZooKeeper* necesario para el funcionamiento de *Apache Kafka*.
- *confluent/cp-kafka* [19] con un servidor *Apache Kafka* sobre el cual se crearán los flujos mediante la incorporación de nuevos *topics*.
- *spark-master* [20] imagen para la creación de un nodo maestro *Spark*.
- *spark-worker* [20] para los nodos esclavos de *Spark*.
- *fis-hubu-producer* es la imagen encargada de crear un servidor *UDP* que reciba el vídeo del paciente y encole los fotogramas a *Kafka*.
- *fis-hubu-consumer* que consume los fotogramas de un flujo de *Kafka* y los procesa utilizando *Spark Streaming*.

En la figura 5.10 se puede observar las conexiones que se darían entre las máquinas y todas están en la misma red virtual.

Las máquinas *fis-hubu* se han creado en exclusiva para este proyecto, funcionan sobre *Ubuntu 18.04* y crean un entorno *conda* para la correcta ejecución del algoritmo. Las modificaciones que fuesen necesarias para adaptar estos contenedores se encuentran en el anexo C.3.

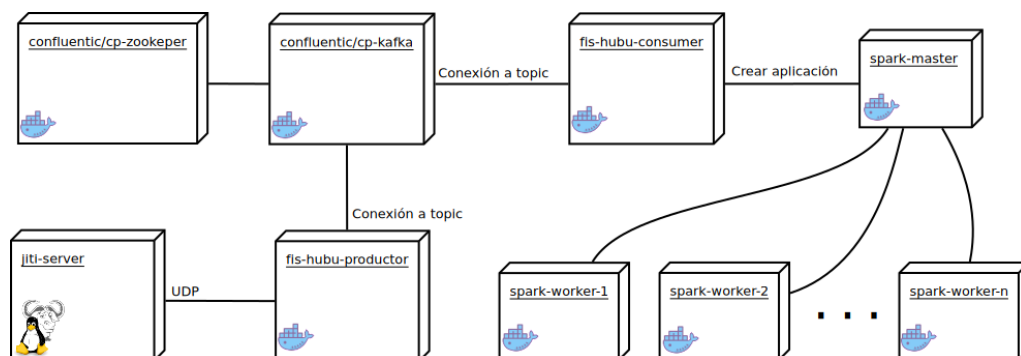


Figura 5.10: Máquinas virtuales *Docker* que implementan el flujo. Se muestra además la conexión con el servicio *Jitsi* que recibe los datos y cómo se incorporan a *fis-hubu-consumer*. Las máquinas *fis-hubu-productor* y *fis-hubu-consumer* son únicas para cada flujo, pero el resto son comunes a todos los flujos paralelos. La cantidad de *workers* que se quisieran lanzar depende de la [Ecuación 5.2](#).

El orden de instancia de cada una de las máquinas es muy importante para que pueda ofrecerse el servicio adecuadamente. Lo primero que se ha de hacer es levantar las máquinas (en este orden) de *Zookeeper*, *Kafka*, nodo maestro de *Spark* y por último los *workers* que se deseen. Es importante asegurar el uso de una red virtual para que todos los *dockers* se localicen entre sí o bien utilizar las conexiones con los puertos que se publiquen, aunque esta decisión no es la más acertada si el servidor puede ser usado con otros motivos.

Un detalle importante a tener en cuenta es que los nodos de *Spark*, tanto el maestro como los esclavos deben tener acceso de alguna manera al código de ejecución, su entorno y sus características. Por tanto se hizo una modificación sobre estas imágenes para que heredasen de la imagen de *fis-hubu-consumer* y así poder ejecutar cada nodo el código de *Python* correctamente.

Una vez las máquinas «servidoras» estén disponibles, se pueden lanzar tanto *fis-hubu-productor* como *fis-hubu-consumer* garantizando un nombre identificativo para que no coincida con otros flujos que puedan aparecer.

Por último se ha de inicializar el *topic* de *Kafka* en su máquina correspondiente, lanzar el *script* de procesamiento en el *fis-hubu-consumer* y el *script* de encolado en el *fis-hubu-productor*.

Para facilitar este proceso se han incorporado varios *scripts* sobre *bash* para automatizar la inicialización de las imágenes *docker* y el lanzamiento de los *scripts* de *Python* necesarios.

El despliegue final se ha realizado utilizando una única máquina, pero el proceso sería similar si las imágenes *docker* estuviesen lanzadas en un clúster de equipos.

5.4. Extensibilidad

Para que el flujo se pueda utilizar para el proceso que se desee sobre los vídeos, se ha desarrollado una forma sencilla para que el programador pueda incrustar su código y se ejecute al final del flujo. Es importante que de utilizar algún objeto de ámbito global en esta extensión (como puede ser un modelo previamente cargado) es necesario que pueda ser serializable. En caso contrario, cuando *Spark* desee enviar el objeto a un nodo esclavo, detendrá el flujo debido a no poder cargar el objeto.

A la hora de extender el algoritmo es importante conocer el tiempo de ejecución necesario para procesar un *frame* y con ello elegir correctamente la cantidad de *workers* de *Spark* necesarios para que no se ralentice la ejecución global. Para elegir la cantidad de *workers* necesarios, se utiliza la ecuación 5.2 que depende del tiempo de procesado medio, su desviación estándar y la tasa de *fps* a procesar.

$$\left\lceil \frac{(t_{mean} + 2t_{std}) * fps}{1000} \right\rceil \quad (5.2)$$

Análisis temporal del flujo

Si se quiere extender el flujo es importante primero conocer el tiempo que necesita de base, lo cual se debe tener en consideración a la hora de incorporar nuevas funcionalidades al flujo y aumentar el número de *workers* para mantener el *status* de tiempo real.

Para analizar el tiempo del flujo, se creó una simulación del mismo que calculase el tiempo de procesado de cada uno de los fotogramas para, posteriormente, calcular la media y la desviación. Concretamente se utilizaron un total de 4863⁸ fotogramas analizados diez veces. Además

⁸Del conjunto de vídeos grabados para pruebas por parte de José Miguel Ramírez Sanz.

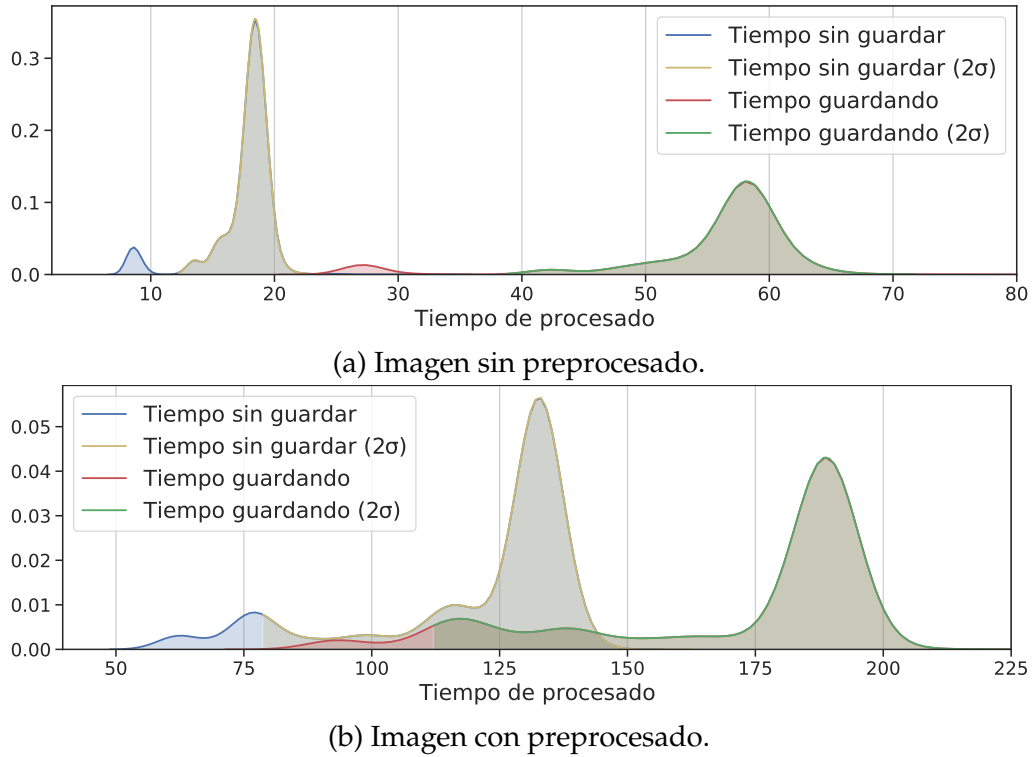


Figura 5.11: Distribución de los datos temporales para el proceso del flujo sobre una imagen.

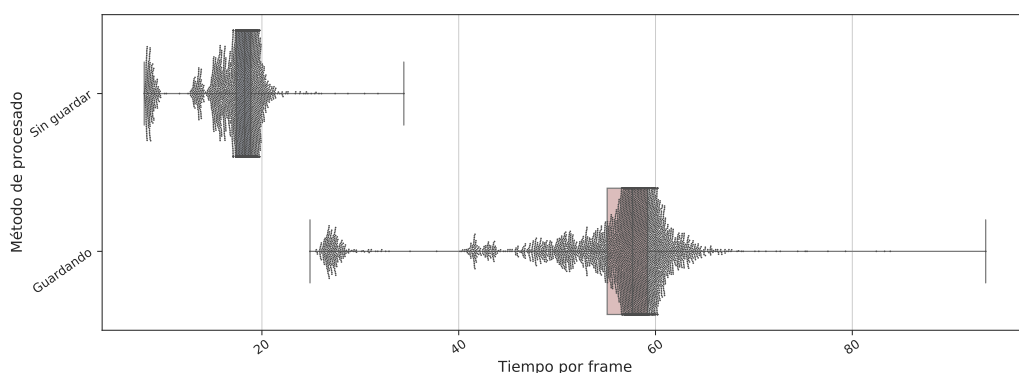
se discriminó entre sin almacenar el fotograma y almacenándolo, y se estudiaron el flujo sin procesar (únicamente la deserialización) o con el preprocesado completo (detección y anonimización del rostro y reparación del brillo y el contraste). Las medidas se calcularon sobre la ejecución de un procesador *Intel Xeon* de 10 núcleos.

Pero antes de pasar a la muestra de los resultados, es importante destacar que el uso de dos desviaciones típicas abarca la suficiente muestra de los datos como para considerarlo un dato adecuado.

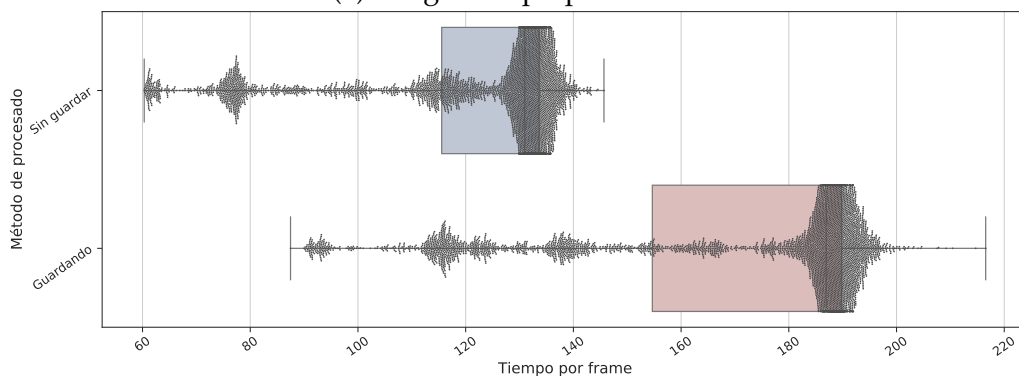
En la figura 5.11 se pueden observar las distribuciones temporales para los casos considerados. La regla de las dos sigmas se cumple siempre con las distribuciones normales, sin embargo, aunque la distribución no es una normal, sí que asemeja su forma y abarca principalmente los datos temporales mayores por lo que se puede garantizar que el usar dos sigmas es suficiente rango temporal para el cálculo de *workers*.

En la figura 5.12 se observa la distribución estadística de los datos. Como es esperable el guardar las imágenes necesita de un tiempo conside-

blemente mayor. Los datos sin procesar la imagen son significativamente bajos, incluso la media sin guardar está en 17,5 milisegundos, lo que alcanzaría un procesado de casi 60 *fps* para cada *worker*. Son métricas muy buenas si se quiere extender el flujo que no necesite la anonimización del rostro (como en cámaras de vigilancia) o que ya vengan contrastadas. En el caso del preprocesado necesita de media 464 milisegundos, que apenas da para procesar dos fotogramas por segundo.



(a) Imagen sin preprocesado.



(b) Imagen con preprocesado.

Figura 5.12: Gráficos de bigotes con la distribución estadística de los datos temporales para el proceso del flujo sobre una imagen.

Con estos resultados, usando la [Ecuación 5.2](#), podemos ver en la [Tabla 5.1](#) la cantidad de *workers* necesarios para ofrecer un procesado en tiempo real.

Tasa de <i>fps</i>	Sin guardar		Guardando	
	Sin procesar	Preprocesado	Sin procesar	Preprocesado
5 <i>fps</i>	1	2	1	3
15 <i>fps</i>	1	1	2	4

Tabla 5.1: N.º de *workers* necesarios según el proceso a realizar.

Integración con el modelo de detección de esqueleto del paciente

Se ha querido comprobar que la arquitectura de colas presentada en este trabajo funcionase correctamente en el proyecto que la engloba. Para ello se ha juntado con el modelo de detección del esqueleto y el cálculo de la diferencia entre dos estados.

Siguiendo las indicaciones⁹ para hacer extensible el algoritmo, es decir, incorporar nuevas funcionalidades se ha incluido dentro del fichero `extraOpt.py` las importaciones de las clases creadas por José Miguel Ramírez Sanz, así como la instancia de la interfaz para acceder al modelo.

El primer detalle a tener en cuenta es que el modelo tenía que instanciarse en el ámbito global de la aplicación para que fuese serializado y enviado a todos los *workers* de *Spark*, ya que necesita para cargar entre 4 y 6 segundos, un tiempo excesivo que se sumaría al análisis de cada fotograma. Sin embargo, la inicialización del flujo necesita más tiempo, por lo que es muy importante lanzarlo lo antes posible para que cuando lleguen los primeros fotogramas esté completamente cargado.

Junto con la dificultad temporal surgió también una dificultad en memoria, concretamente la memoria de la tarjeta gráfica. Cada instancia cargada del modelo reservaba entre 512 y 768 MiB de VRAM por lo que según la cantidad de *workers* se necesita una mayor cantidad de memoria en el sistema. En el caso del equipo personal, que se contaba con una *Nvidia 750 Ti* de 2 GiB de memoria, solamente se podían ejecutar un *worker* junto con el *master*, teniendo en cuenta la ecuación 5.2 implicaría que se podían analizar 12 *fps* en el caso de utilizar el procesado básico y 3 *fps* en el caso de ejecutar el flujo completo. Teniendo en cuenta las necesidades del tiempo real este dispositivo no es adecuado.

⁹Ver [Sección C.3](#).

Al utilizar las GPU de la máquina *gamma* del grupo ADMIRABLE, que cuenta con tres *Nvidia Titan Xp* de 12 GiB de memoria cada una, se podían llegar a utilizar un total de 47 *workers* junto con el nodo *master*. Permitiendo así unas tasas de *fps* de 288 en el caso de solo usar el procesamiento básico y de 73 con el flujo completo. Si los vídeos se mantienen con los 15 *fps* se pueden tener 4 flujos simultáneos y en el caso de reducir la tasa a 5 se pueden tener hasta 14 flujos.

Análisis temporal del flujo con toda la integración

Tal como se muestra en la [Sección 5.4](#) se realiza el mismo experimento con el flujo completo, es decir, se incorpora al preprocesado el modelo creado para este proyecto. En la [Figura 5.13](#) se puede observar la distribución estadística de los datos evaluados. La media sin guardar la imagen es de 464 ms y guardando de 626 ms, las desviaciones típicas son de 86,5 y 98,7 respectivamente.

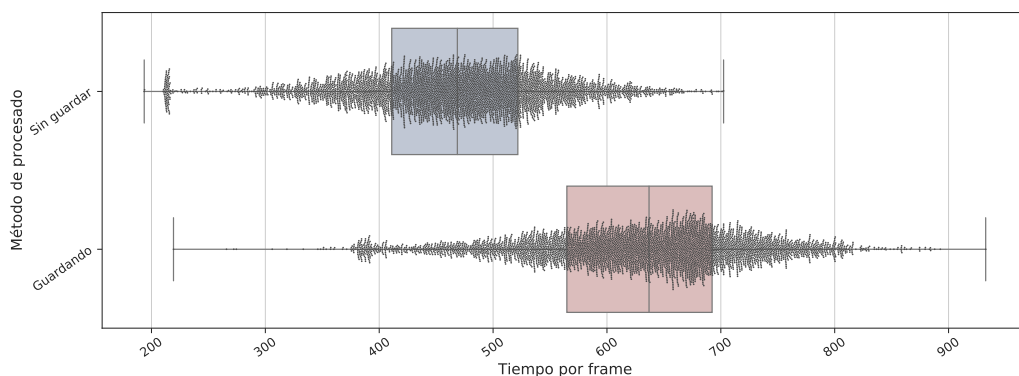


Figura 5.13: Gráficos de bigotes con la distribución estadística de los datos temporales para el proceso del flujo completo sobre una imagen.

Se necesitan, utilizando 15 *fps* al menos 10 *workers* si no se guarda y 13 en caso contrario. Por otro lado, de utilizar 5 *fps* se necesitan 4 y 5 *workers* sin guardar y guardando los fotogramas respectivamente.

Trabajos relacionados

Existe una literatura variada sobre técnicas para la rehabilitación de pacientes de la enfermedad del *Parkinson* así como de aproximaciones para el análisis de vídeo en tiempo real. En este capítulo se expondrán algunos de los más relevantes.

6.1. Literatura científica relacionada

En el año 2019, Linares-del Rey *et. al.* [22] realizaron un meta-análisis sobre un total de veintiséis artículos de aplicaciones móviles para pacientes de *Parkinson*. Se consideraron artículos en castellano y en inglés entre los años 2011 y 2016. Además se analizaron ciento tres aplicaciones móviles que estuviesen en los principales *marketplaces*: *Google Play*, *App Store* y *Windows Store*.

El conjunto de estudios abarcaba a un total de cuatrocientos veinte pacientes y docientos treinta y dos personas sanas, estos como grupo de control en los artículos que los incluían.

Para valorar la calidad metodológica de estos artículos se utilizó la escala JADAD [17] basada en si existe o no aleatorización de los participantes, y se describe el método, si se hace un estudio de doble ciego, y se describe el método llevado para ello, y si se describen adecuadamente las pérdidas. Se suele tomar como valor mínimo aceptable una puntuación de tres.

Entre el total de artículos revisados el único con una puntuación aceptable fue el sistema *CuPid* [15] con un total de cuarenta participantes. La aplicación que presenta evaluaba la mejora de la calidad de vida del

paciente según mejorasen su equilibrio, marcha y resistencia. Sin embargo, esta aplicación requería de uso de sensores externos al dispositivo móvil para poder evaluar la aplicación.

Respecto a las aplicaciones, al no tener artículos asociados (ya que de tenerlos se habrían analizado en el primer conjunto) no se podían evaluar siguiendo la misma escala. Sin embargo, algunas aplicaciones de las que se analizaron ya estaban incluidas en otras revisiones similares a las de Linares-del Rey *et. al.*

La conclusión de los autores fue que debido a la baja calidad metodológica y la poca cantidad de estudios realizados, no se podía recomendar un uso generalizado de las aplicaciones, es decir, sigue haciendo falta la supervisión de terapeutas especializados que puedan dirigir y ayudar a la mejora de la calidad de vida de los pacientes.

6.2. Aproximaciones en problemas similares

El arquitecto de software Amit Baghel presentó en 2016 una aproximación en Java para procesar vídeo utilizando *Kafka* junto a *Spark* [9].

El objetivo de su trabajo era la creación de un detector de movimiento que combinase en una misma cola de *Kafka* diferentes fuentes de vídeo y las procesase utilizando *OpenCV* junto a *Spark Streaming* y comparar cada *frame* con el anterior y así detectar los objetos que se hubiesen movido.

Debido a la similitud de la arquitectura de este trabajo con la que se deseaba contruir se utilizó como base en algunas decisiones del diseño. Concretamente se decidió mantener la configuración planteada para *Kafka*: cantidad de particiones en tres y el factor de replicación de uno.

Sin embargo, la arquitectura presentada, debido a los cambios de las versiones de los últimos años, ya no se puede desplegar en los sistemas *GNU/Linux* modernos a menos que se compilen directamente las versiones que plantean.

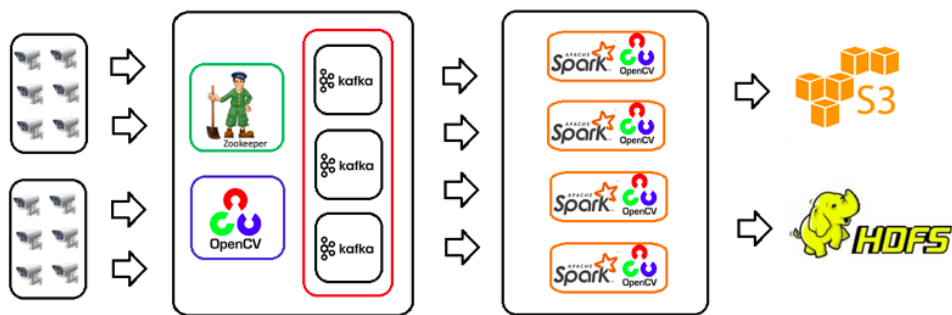


Figura 6.14: Diagrama de la arquitectura presentada por Amit Baghel.
Fuente: [Infoq](#)

Conclusiones y Líneas de trabajo futuras

Para finalizar la exposición de esta memoria se comentará en este capítulo las conclusiones del trabajo así como líneas futuras que se van a realizar.

7.1. Conclusiones

De este trabajo se han sacado diversas conclusiones:

- Las herramientas que ofrece la *suite Apache* para la gestión y procesado de flujo son muy robustas y tiene el desempeño que se espera de ellas, especialmente referente a la flexibilidad y fiabilidad. Sin embargo, existen aún muchas limitaciones en la documentación que en la mayoría de las ocasiones es muy «esotéricas», especialmente a la hora de enseñar a integrar las diversas tecnologías. De hecho, una de las mayores dificultades encontradas para el cumplimiento de los objetivos de este TFM ha sido la integración correcta de las herramientas.
- Los objetivos del proyecto han sido cumplidos al completo, habiendo conseguido tanto una aplicación sencilla para comunicar a pacientes y usuarios como también una arquitectura fundada sobre una tecnología robusta para facilitar el análisis en tiempo real. Se espera que esto conlleve una mejora continua de la calidad de vida de los pacientes de la enfermedad de *Parkinson* al tener un acceso más fácil a los profesionales de la terapia ocupacional.

7.2. Líneas futuras

Dentro de las líneas futuras están:

- Integrar el flujo con la aplicación de *Jitsi* para que funcione automáticamente con los pacientes.
- Desplegar su uso con terapeutas y pacientes reales a una mayor escala. Actualmente se tienen tres pacientes y un terapeuta, pero el objetivo a largo plazo es que la herramienta se pueda usar por una cantidad mayor de personas.
- Automatizar el despliegue en clúster de servidores utilizando *Kubernetes* [8], herramienta creada por *Google* que permite la creación fácil de un clúster de máquinas virtuales *Docker* lo que permitiría una mayor versatilidad sobre los *scripts* creados en este trabajo.
- Explorar mejores configuraciones para *Kafka* y *Spark Streaming* para mejorar el desempeño del flujo.

Apéndices

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apéndice se expondrán los distintos *sprints* que se han realizado y un estudio de la viabilidad del proyecto.

A.2. Planificación temporal

La planificación temporal se ha realizado adaptando la metodología *Scrum*. Para poder adaptarlo a un trabajo de una sola persona para un proyecto educativo se han considerado las siguientes indicaciones:

- El desarrollo se ha basado en iteraciones o *sprints* de dos semana de duración aproximadamente.
- Cada uno de los *sprints* contiene las tareas que se realizaron en el mismo.
- Cada tarea tiene un coste estimado dependiendo de lo que el programador estime conveniente siguiendo los parámetros de tiempo a emplear, dificultad técnica entre otros.
- Una vez concluida una tarea se especifica el coste real para poder estimar de una manera más correcta tareas de *sprints* siguientes.
- Al finalizar cada *sprint* se realiza una reunión con los tutores del proyecto.

Sprint 0

El *sprint* 0 consistió en el desarrollo de la aplicación web para la recogida de datos para el proyecto. Es el único *sprint* realizado en colaboración con el alumno José Miguel Ramírez Sanz.

Tarea	Est.	Final
Diseño de la interfaz web	3	3
Creación de la plantilla maestra de toda la web	3	2
Creación de la plantilla base para el menú del paciente	5	5
Implementación del menú del terapeuta	2	2
Creación de la conexión para videollamada - Terapeuta	1	1
Creación de la conexión para videollamada - Paciente	1	1
Implementación del sistema de inicio de sesión	2	3
Creación de <i>plugin</i> para la captura y emisión del vídeo del paciente	8	13
Creación de la interfaz de gestión del paciente	2	2

Tabla A.1: Tareas del *sprint* 0

La mayor dificultad en este *sprint* fue la creación del *plugin* sobre *Jitsi* que duplicase el flujo de vídeo y emitiese los datos del servidor al sistema de colas. Esto fue debido a la poca documentación de *Jitsi* sobre la implantación de estas modificaciones.

Sprint 1

El *sprint* 1 consistió en la exploración de herramientas para la creación y procesado de flujos de vídeo.

Tarea	Est.	Final
Búsqueda de herramientas para la creación de flujos de datos	3	3
Pruebas sobre <i>Apache Flume</i>	5	3
Pruebas sobre <i>Apache Kafka</i>	5	5
Búsqueda de herramientas para el despliegue por contenedores	3	3
Prueba de despliegue de <i>Apache Kafka</i> para <i>Docker</i>	3	3
Búsqueda de herramientas para el procesamiento de flujos de vídeo	3	3
Pruebas con <i>Spark Streaming</i> con <i>OpenCV</i>	5	13

Tabla A.2: Tareas del *sprint* 1

El desarrollo de este *sprint* fue bastante similar a lo planificado. El caso particular estuvo en la última tarea debido a la complejidad conectar y procesar de manera efectiva *Spark Streaming* con la librería de *OpenCV* y poder garantizar que no se perdían *frames*.

Sprint 2

El *sprint* 2 consistió en la implementación de las conexiones entre todos los elementos del flujo a nivel local.

Tarea	Est.	Final
Despliegue local de <i>Apache Spark</i>	2	2
Despliegue local de <i>Apache Kafka</i>	3	5
Simulación de servidor UDP para vídeo	5	5
Ingestor a Kafka del vídeo UDP	3	5
Conectar <i>Spark Streaming</i> con <i>Kafka</i>	3	13
Implementar un anonimizador de rostros	3	2
Parametrizar todos los <i>scripts</i> creados	2	2

Tabla A.3: Tareas del *sprint* 2

La razón de la gran diferencia en la quinta tarea del *sprint* entre predicho e invertido fue debido a que la documentación de *Kafka* no estaba bien detallada para conectar con *Spark Streaming*.

Sprint 3

El *sprint* 3 consistió en la implementación de una infraestructura de contenedores *Docker* que conectase todos los servicios para el flujo.

Tarea	Est.	Final
Despliegue de <i>Apache Spark</i> para el nodo <i>master</i>	2	2
Despliegue de <i>Apache Spark</i> para varios nodos <i>worker</i>	2	1
Creación de imágenes <i>Docker</i> para soportar las aplicaciones creadas	8	21
Despliegue de la aplicación <i>openCV</i>	3	3
Recogida de <i>stream</i> de vídeo e ingestión en <i>Kafka</i> en <i>Docker</i>	8	8

Tabla A.4: Tareas del *sprint* 3

Este *sprint* fue más sencillo que el anterior, principalmente porque en el anterior existieron dificultades para cumplir con los plazos del mismo al complicarse la tarea de conectar dos de los componentes. Con la experiencia obtenida en ese *sprint* algunas tareas se desarrollaron más fácilmente. Por otro lado el despliegue con *docker* necesitó casi un *sprint* aparte ya que muchas de las operaciones, *a priori* simples, resultaron tener varias capas de complejidad no previstas. Esto causó el mayor desajuste entre lo previsto y lo invertido de todo el proyecto, presumiblemente por la poca experiencia previa en el uso de contenedores *docker*.

Sprint 4

El *sprint* 4 consistió en la automatización de los procesos del *sprint* 3.

Tarea	Est.	Final
<i>Script</i> para la creación de <i>topics</i> de <i>Kafka</i>	1	1
<i>Script</i> de lanzamiento de instancia del flujo	8	13
<i>Script</i> de inicialización de los servicios	2	2
<i>Script</i> para la eliminación de un <i>topic</i> de <i>Kafka</i>	1	1
<i>Script</i> para el cierre ordenado de un flujo	5	8
<i>Script</i> para el apagado completo de los servicios	1	1

Tabla A.5: Tareas del *sprint* 4

La mayor dificultad de este *sprint* estuvo en la creación de los *scripts* que abarcasen tanto el inicio como el cierre de un flujo debido a que estaban involucrados tanto el inicio de las imágenes *docker* asociadas, el control de identificadores (realizado mediante semáforos en forma de ficheros) y la ejecución ordenada. Por estos motivos las predicciones no fueron acertadas al complicarse la implementación.

Sprint 5

El *sprint* 5 consistió en la creación de experimentos para evaluar el tiempo necesitado para el procesado del flujo además de la búsqueda de un sistema de serialización y compresión adecuado para el encolado de los fotogramas.

Tarea	Est.	Final
Diseño de los experimentos	3	3
Implementación de los experimentos del flujo	5	13
Implementación de los experimentos sobre las técnicas de serialización y compresión	5	5
Implementación de la recogida y visualización de los resultados	3	3
Análisis de los resultados	3	3

Tabla A.6: Tareas del *sprint* 5

Hubo una gran dificultad a la hora de la implementación de los experimentos debido a que se quiso paralelizar la ejecución y acelerar así el proceso. Sin embargo, esto conllevó un estudio más en profundidad de las herramientas de *Python* para ello.

Sprint 6

En este penúltimo *sprint* se comenzó el despliegue real y la integración con el modelo predictivo de poses del alumno José Miguel Ramírez Sanz.

Tarea	Est.	Final
Implementación de la extensibilidad del flujo	2	2
Instalación local del flujo completo	3	3
Instalación de las librerías necesarias en el equipo <i>Gamma</i>	1	1
Despliegue de las máquinas <i>docker</i> en <i>Gamma</i>	1	1
Pruebas sobre vídeos pregrabados	2	2
Adaptación de los <i>dockers</i> para compatibilidad con <i>Nvidia</i>	8	13
Diseño de experimentos sobre el flujo completo	1	1
Implementación de los experimentos	8	5
Análisis de los resultados	2	2

Tabla A.7: Tareas del *sprint 6*

El *sprint* se desarrolló con bastante facilidad respecto a otros debido principalmente a que gran parte del material necesario, como los *scripts* de lanzamiento o experimentos similares, ya se habían creado anteriormente. El único problema estuvo relacionado con la adaptación de las imágenes *Docker* a *Nvidia* ya que se requirieron muchos cambios inesperados para que funcionase adecuadamente.

Sprint 7

Este último *sprint* consistió en la creación de la documentación del trabajo realizado.

Tarea	Est.	Final
Maquetación de la plantilla	1	1
Escritura de la introducción	2	2
Escritura de los objetivos	3	3
Escritura de los conceptos teóricos	5	5
Escritura de los aspectos relevantes	13	13
Escritura de los trabajos relacionados	3	3
Escritura de las conclusiones y líneas futuras	2	2
Escritura del plan de proyecto	2	2
Escritura del diseño	5	5
Escritura del manual del programador	3	3
Creación de la presentación	8	8

Tabla A.8: Tareas del *sprint 7*

A.3. Estudio de viabilidad

Viabilidad económica

Debido a que este TFM está dentro de un proyecto donde también se encuentra el TFM de José Miguel Ramírez Sanz, se ha realizado el estudio de viabilidad económica de manera conjunta

En la [Tabla A.9](#) se encuentran los costes total en salarios en jornada completa durante seis meses para dos empleados.

Concepto	Coste (€)
Salario mensual bruto [16]	2 047,78
Seguridad Social (30,04 %)	615,15
Retención IRPF (2 %)	28,65
Salario mensual neto	1 403,97
Total 6 meses y dos empleados	24 573,36

Tabla A.9: Costes de personal.

La [Tabla A.10](#) están las inversiones y amortizaciones en materia de *hardware*, tanto los *MainFrames* para el despliegue y el cálculo como los equipos de desarrollo.

Concepto	Coste (€)	Coste amortizado (€)
Ordenador de desarrollo (x2)	950	59,37
Dispositivos pacientes (x9)	100	6,25
Webcam pacientes (x9)	150	9,38
MainFrame Gamma	3 000	187,5
Gamma GPU (x3)	1 500	93,75
MainFrame Alpha	2 000	125
Total	13 650	853,16

Tabla A.10: Costes de *hardware*.

Por último en la [Tabla A.11](#) se encuentran los dos servicios contratados para dar acceso a la aplicación a los pacientes.

Servicio	Coste (€)
Suscripción <i>Ngrok</i>	7,33
Lineas <i>Vodafone</i> (x4)	30
Total (por 6 meses)	763,68

Tabla A.11: Costes de servicios.

Los costes totales se pueden observar en la [Tabla A.12](#).

Servicio	Coste (€)
Costes de personal	24 573,36
Costes de <i>hardware</i>	13 650
Costes de servicios	763,68
Total	38 987,04

Tabla A.12: Costes totales.

Viabilidad legal

Este proyecto se ha realizado con la ayuda de software de terceros con licencias propias que influyen sobre la viabilidad legal del proyecto.

Dentro de las licencias de las herramientas utilizadas están:

- **MIT:** Esta licencia permite el uso comercial del producto, la modificación del mismo, la libre distribución y el uso privado. No tiene garantías ni responsabilidad. La única condición es hacer referencia a ella. Como no obliga a mantener la licencia ni afecta a la distribución del software que use la licencia final del producto, esta puede ser cualquiera.
- **GPLv3:** Licencia que permite uso comercial, distribución, modificación uso privado y creación de patentes. Obliga a licenciar cualquier modificación del código o códigos que usen herramientas con esta licencia usar **GLPv3** u otras versiones.
- **Apache 2.0:** Licencia de las herramientas de Apache, tiene las mismas propiedades que la licencia GPLv3 con excepción de que no obliga a que las nuevas implementaciones con dependencias en Apache 2.0 sean licenciadas como código libre.

- **BSD:** Tiene características semejantes a MIT en el contexto en el que la usamos.
- **BSD 3-Clause:** Tiene características semejantes a MIT en el contexto en el que la usamos.

La licencia más restrictiva del proyecto, la *GPLv3*, está asociada a las plantillas de imágenes *docker* de Mario Juez Gil [20]. Como el resto de licencias son compatibles con la *GPLv3* se utiliza dicha licencia para todo el proyecto.

Copyright de terceros

Apache 2.0:

- *Apache Kafka - Apache Foundation*
- *Apache Zookeeper - Apache Foundation*
- *Apache Spark - Apache Foundation*
- *Docker CP - Confluentic*
- *Jitsi Meet - Jitsi*

GPLv3:

- *Clúster Spark Docker - Mario Juez Gil*

BSD todas sus variantes:

- *Caffe - BVLC*
- *Flask - Pallets*
- *Jinja - Pallets*
- *OpenCV - Intel Corporation, Xperience AI*
- *Seaborn - Michael Waskom*

MIT

- *Bootstrap 4 - Twitter*
- *jQuery - JS Foundation*

Apéndice B

Especificación de diseño

B.1. Introducción

El sistema completo funciona con la unión de dos subsistemas. El subsistema web, que lo forman el servidor que identifica al usuario, y la plataforma *Jitsi*, para realizar la videollamada y el subsistema de colas, que da nombre a este trabajo, que encola los *frames* del vídeo y los procesa.

En este apartado se explica cómo se ha diseñado el sistema completo, la comunicación entre todas sus partes, el flujo del trabajo y la composición de cada elemento.

B.2. Diseño procedimental

El procedimiento para el procesado de los vídeos ([Figura B.1](#)) para cada paciente que entre en una videollamada (independientemente de que esté o no el terapeuta) se inicializa un nuevo Ingestor que encolará los frames que reciba. A su vez este Ingestor creará un procesador que consumirá los *frames* y los procesará. Estos dos elementos permanecerán hasta que sean cerrados por la finalización de la conexión. Por tanto, por cada paciente conectado se procesará tan pronto como sea posible la secuencia de *frames* del mismo.

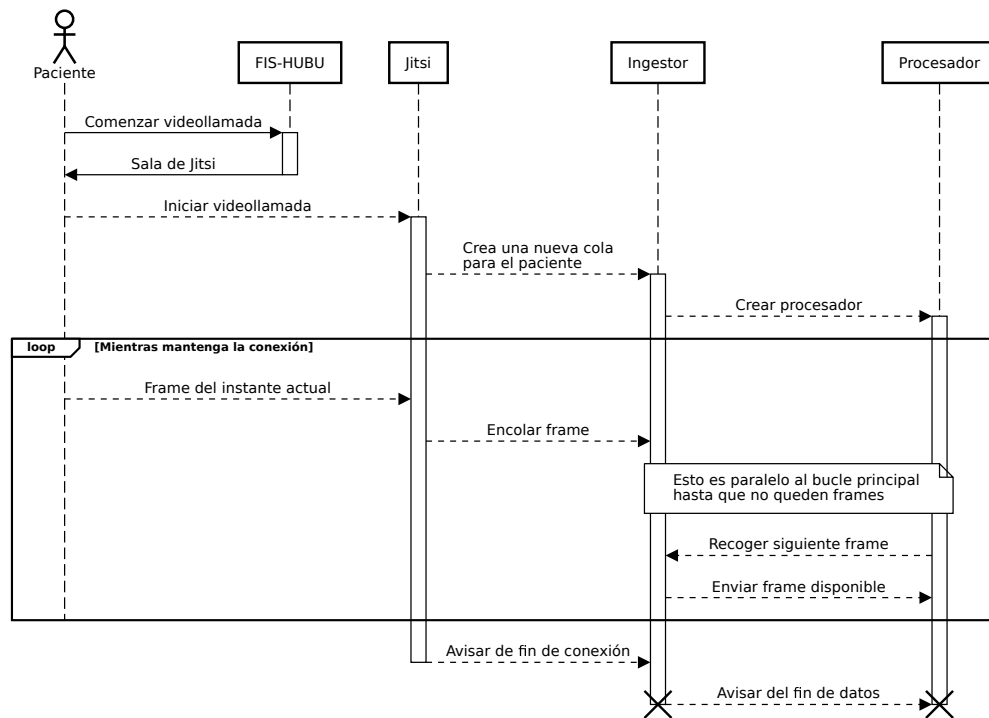


Figura B.1: Diagrama de secuencias para el proceso general de la aplicación. Conexión del paciente y procesado del vídeo de entrada.

B.3. Diseño arquitectónico

La parte más esencial de este trabajo es la creación del *pipeline* que procesa en tiempo real los *frames* de la comunicación del paciente. Todo ello se ha desarrollado, utilizando las herramientas de la suite de *Apache*, *Apache Kafka* para el *ingestor*, *Apache Spark Streaming* para el *procesador*. En la [Figura B.2](#) se puede observar el funcionamiento completo del flujo y las partes que lo componen.

Para que esta arquitectura sea ajena al entorno, se encapsulan sobre máquinas virtuales *Docker* de tal forma que se pueda desplegar en cualquier sitio. Concretamente hay cuatro tipos de imágenes *Docker*. La primera se encarga de la serialización de los frames y lanzan el *script* de *Python* de encolado, esta máquina se crea y destruye a voluntad de las conexiones de los pacientes. La segunda y tercera imágenes son el servicio de *Zookeeper* y de *Kafka*. Estas imágenes no se duplican en caso de cambios en las conexiones, únicamente se crean o borran colas. Por último, la cuarta imagen es la transformación del flujo. Se crean o se destruyen según

las conexiones de los pacientes y se parametrizan para que consuman un flujo concreto y hagan un procesamiento concreto. De esta manera, diferentes pacientes se podrían configurar para tener diferentes procesados permitiendo una mayor flexibilidad en el procesamiento.

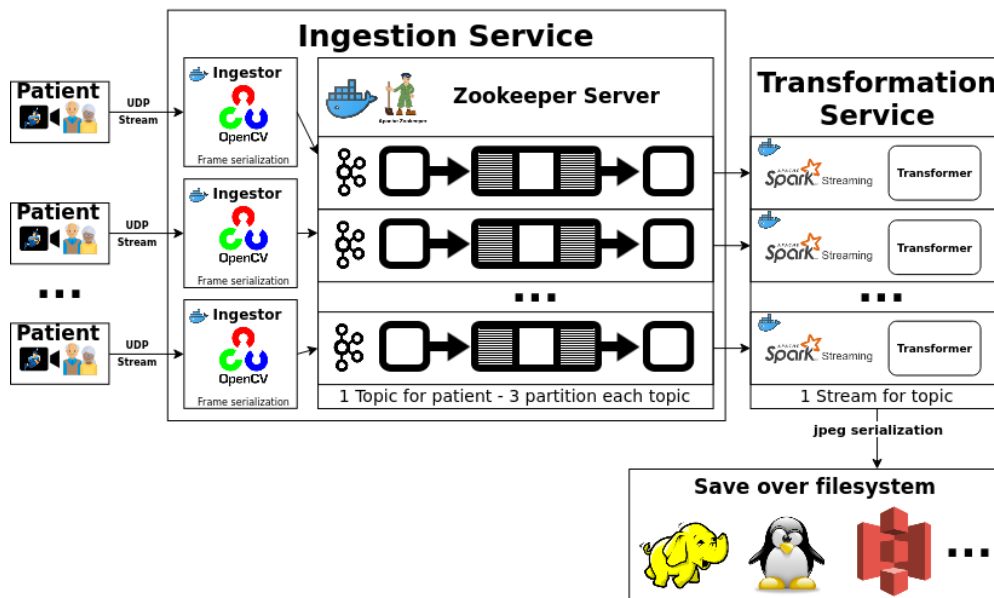


Figura B.2: Implementación del flujo ETL para el procesamiento de imágenes en tiempo real. Por cada paciente conectado se asocia a un *ingestor* que serializa la imagen y la encola en un flujo de *Apache Kafka* dividido en tres particiones. Cada cola es procesada por un servidor de *Apache Zookeeper*. El flujo es consumido por un procesador de *Apache Spark Streaming*. El procesamiento puede ser cualquier operación con imágenes. Por último se almacena en el sistema de ficheros los resultados del flujo.

El diagrama de despliegue de las máquinas virtuales *docker* se puede observar en la [Figura B.3](#).

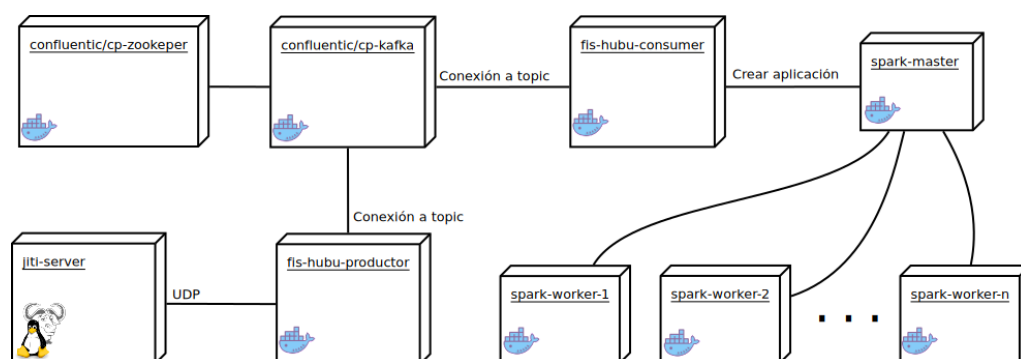


Figura B.3: Diagrama de despliegue de las máquinas virtuales *docker*.

Apéndice C

Documentación técnica de programación

C.1. Introducción

El objetivo de esta sección es documentar a nivel técnico los componentes del proyecto para facilitar la extensión, modificación y uso por parte de los programadores.

C.2. Estructura de directorios

En la carpeta `doc` se encuentra la documentación del proyecto. En `src` los códigos fuentes para el proyecto.

Dentro de los fuentes están la carpeta `dockers` donde se incorporan tanto los `Dockerfile` como los `docker-compose.yml` que contienen la información de los contenedores que darán soporte a la aplicación. En `process` están los fuentes *Python* para el emisor de fotogramas, el productor y el consumidor. En la carpeta `scripts` y en sus subcarpetas `deploy` y `helpers` están todos los *scripts* sobre *Bash* necesarios para el despliegue de las imágenes *docker* y las funciones que deben ejecutar, concretamente en `deploy` se encuentran los programas para la creación y cierre de imágenes y en `helpers` las órdenes sobre las aplicaciones que contienen las imágenes. Por último en `tools` se encuentran herramientas y *notebooks* de *jupyter* que se han desarrollado para desarrollar este trabajo.

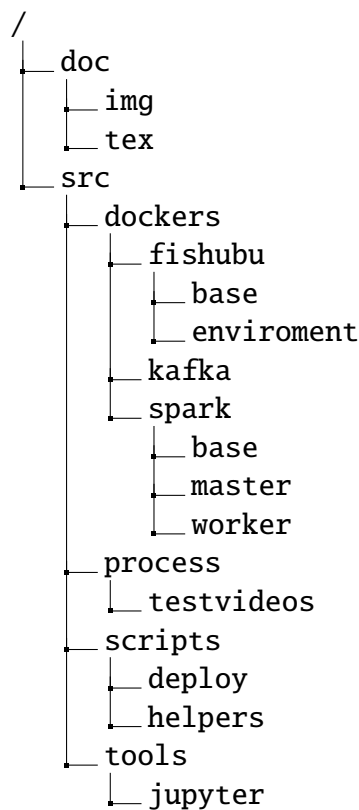


Figura C.1: Árbol de directorios

C.3. Manual del programador

Scripts Python

Se han creado tres *scripts* de *Python* para ofrecer el servicio del sistema de colas. Por el desarrollo que se ha llevado, funcionan tanto dentro como fuera de una imagen *Docker*. Si se fuesen a usar fuera a nivel local, el uso sería el siguiente:

emitter.py

Script encargado en enviar a un servidor UDP fotogramas de vídeos. La ayuda:

```

1 Sintaxis:
2   emitter.py --ip=localhost --port=12345 --file=video.webm
3   -----

```



```

4 Parámetros de comunicación
5   --ip=<IP de emisión>
6     (Por defecto: localhost)
7   --port=<Puerto>
8   --file=<Fuente de video>
9
10 Parámetros de gestión del flujo
11   --resize=<Proporcion>
12     (Por defecto: 1.0)
13   -f <FPS> | --fps=<FPS>
14     (Por defecto: 15) tasa de frames del vídeo a emitir

```

producer.py

Script encargado de la ingestión de los fotogramas a *Kafka*. La ayuda:

```

1 Sintaxis
2   producer.py --ip=localhost --port=12345 --topic=queue
3   -----
4 Parámetros de comunicación
5   --ip=<IP de emisión>
6     (Por defecto: localhost)
7   --port=<Puerto>
8   --kafkahost=<Dirección de kafka>
9     (Por defecto: localhost:9092)
10  --topic=<Topic de Kafka>
11    (Por defecto: video-stream-event)

```

consumer.py

Script encargado del procesado en paralelo y tiempo real del vídeo. La ayuda:

```

1 Sintaxis
2   consumer.py --ip=localhost --port=12345 --topic=queue
3   -----
4 Parámetros de comunicación
5   --output==<Carpeta de salida>
6     (Por defecto: output)
7   --sparkhost=<Dirección de spark>

```

```

8      (Por defecto: local)
9      --kafkahost=<Dirección de kafka>
10     (Por defecto: localhost:9092)
11     --topic=<Topic de Kafka>
12     (Por defecto: video-stream-event)
13
14 Parámetros de gestión del flujo
15 -a -> Anonimizar rostros, por defecto pixalado
16     -g <Factor> | --blur=<Factor>
17     (Por defecto: 3) Anonimizar con blur
18     -p <Factor> | --pixel=<Factor>
19     (Por defecto: 15) Anonimizar con pixelado
20 -b -> Auto ajustar brillo
21 -c -> Auto ajustar contraste
22 -f <FPS> | --fps=<FPS>
23     (Por defecto: 15) tasa de frames del vídeo a emitir
24 --no-save -> No guardar los frames

```

Script de despliegue

Para el despliegue de los servicios mediante *Docker* se han creado cuatro *scripts* encontrados en la carpeta `src/scripts/deploy` (en adelante `deploy`).

Estos códigos en *Bash* son los siguientes:

start-server

Se encarga de instanciar los diferentes servicios.

```

1 Sintaxis
2 start-server <Carpeta de salida> <N. de CPU master> <N.
    de workers> <N. de CPU por worker> <Memoria por
    worker>

```

stop-server

Detiene todos los servicios.

```
1 Sintaxis
2 stop-server
```

new-stream

Genera un nuevo flujo completo que se va a procesar. El funcionamiento es transparente. Recibe los parámetros para el *emitter.py* y para el *consumer.py*. Por seguridad es preferible solamente modificar los parámetros de gestión del flujo y no los de comunicación.

```
1 Sintaxis
2 new-stream "Parámetros_emitter.py" "Parámetros_
   consumer.py"
3 # Es muy importante mantener las comillas
```

Devuelve el identificador del flujo. Es importante este valor para poder cerrarlo después.

stop-stream

Detiene todos los procesos sobre un flujo concreto.

```
1 Sintaxis
2 stop-stream <ID del flujo>
```

C.4. Compilación, instalación y ejecución del proyecto

Como se ha mencionado anteriormente, en la carpeta *deploy* se encuentran los *scripts* para instalar el proyecto y poder ejecutarlo.

Para que los *scripts* se ejecuten correctamente es necesario que se ejecuten sobre un sistema operativo *GNU/Linux* con el servicio de *Docker*

y la extensión *Nvidia container toolkit* [7] instalados. Adicionalmente se necesitará que la máquina donde se vayan a ejecutar los *workers* disponga una tarjeta gráfica *Nvidia* instalada con soporte para *CUDA* 10.2 (para que la integración con el proyecto del compañero José Miguel Ramírez Sanz sea ejecutable). Es posible que sea necesario cambiar el fichero *Dockerfile* de la carpeta *dockers/fishubu/base* para que use los *drivers* de la tarjeta gráfica del equipo sobre el que se lance la imagen *docker*.

Antes de lanzar los servicios es importante haber creado previamente las imágenes maestras para los diferentes *dockers*. Desde la carpeta *deploy*:

```
1 docker build -f ../../dockers/fishubu/base/Dockerfile -t
   fishubu-base:1.0.0 ../../
2 docker build -f
   ../../dockers/fishubu/enviroment/Dockerfile -t
   fishubu-env:1.0.0 ../../
3 docker build ../../dockers/spark/base -t
   spark-base-fis:2.4.5
4 docker build ../../dockers/spark/master -t
   spark-master-fis:2.4.5
5 docker build ../../dockers/spark/worker -t
   spark-worker-fis:2.4.5
```

El orden de ejecución de los *scripts* es el siguiente:

1. Ejecutar **start-server** para que los servicios de *Kafka* y *Spark* estén activos y den soporte a los flujos que lo necesiten.
2. Ejecutar **new-stream** recibiendo como parámetros la configuración deseada para el emisor y para el consumidor ([Sección C.3](#)). Este devuelve el identificador del flujo, será necesario para pedir el cierre del flujo.

Para detener el flujo los pasos son los siguientes:

1. Ejecutar **stop-stream** para cada flujo arrancado.
2. Ejecutar **stop-server** y se detendrán todos los servicios.

C.5. Fallos y soluciones

Los fallos que puedan ocurrir durante la ejecución de los *scripts* dejan varios ficheros de log en la carpeta `/tmp/fishbulogs` así como en la carpeta logs de cada máquina *docker* ejecutada. En caso de fallo del flujo es importante revisar los ficheros para conocer los posibles orígenes. Los fallos más comunes y sus correspondientes soluciones son:

Fallo de memoria

Si la caída del flujo deja una excepción del tipo «`Caused by: java.io.EOFException`» implica que los datos que debe procesar el flujo son mayores que los que caben en memoria.

Para solucionar esto es necesario dar más memoria a cada *worker*. El valor recomendado es de 2 GiB por cada núcleo de *worker*.

Los fotogramas no se procesan

Si los fotogramas no se procesan y observa la existencia del fichero de log «`notprocesslog`» en los *workers*, significa que ha existido un error mientras se procesaba el fotograma. En el mismo log se encuentra el origen del fallo y probablemente se deberá a que el fichero de `extraOpt.py` incluye información errónea o una ruta incorrecta.

Para solucionar esto se debe volver a cargar el fichero `extraOpt.py` y por tanto se han de volver a construir (orden `build`) las imágenes desde la *fishubu-env*. En el caso de que el error persista, usar el flag `--no-cache` a la hora de reconstruir las imágenes.

Error de versión de *Nvidia*

Si a la hora de ejecutar el flujo, en el arranque ocurre una excepción del tipo «`forward compatibility was attempted`» indica que la versión instalada de *Nvidia* sobre *docker* no es compatible con la versión instalada en el equipo.

Para solucionarlo es necesario cambiar el fichero `Dockerfile` de la carpeta `src/dockers/fishubu/base` y cambiar la versión que se instala por la que se tiene en el equipo. Es necesario que al menos sea la versión 440.

Error con la conexión a *Nvidia*

Si el flujo al iniciar da el error «could not select device driver» significa que la extensión de *docker* para la compatibilidad con *Nvidia* no está instalada o no se ha reiniciado el servicio de *docker* desde que se instaló.

Se soluciona instalando la extensión de *docker* «*docker-nvidia*» y reiniciando el servicio.

Bibliografía

- [1] Apache Flume. <https://flume.apache.org/>.
- [2] Apache Hadoop. <https://hadoop.apache.org/>.
- [3] Apache Kafka. <https://kafka.apache.org/>.
- [4] Spark Streaming - Spark 2.4.5 Documentation. <https://spark.apache.org/docs/latest/streaming-programming-guide.html>.
- [5] Github repositories of jitsi. <https://github.com/jitsi>, 2020.
- [6] Mark Adler and Jean-loup Gailly. Gzip. <https://www.gnu.org/software/gzip/manual/gzip.html>, 2018.
- [7] ArchLinux Wiki Authors. Docker. <https://wiki.archlinux.org/index.php?title=Docker&oldid=622596>, 2020. [Internet; descargado 31-junio-2020].
- [8] Los autores de Kubernetes. Documentación de kubernetes | kubernetes. <https://kubernetes.io/es/docs/home/>, apr 2020.
- [9] Amit Baghel. Video stream analytics using opencv, kafka and spark technologies. <https://www.infoq.com/articles/video-stream-analytics-opencv/>, 2017.
- [10] Joachim Bauch. Pylzma. <https://www.joachim-bauch.de/projects/pylzma/>, 2019.
- [11] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine learning for data streams: with practical examples in MOA*. MIT Press, 2018.

- [12] Thomas Boutell and T Lane. Png (portable network graphics) specification version 1.0. *Network Working Group*, pages 1–102, 1997.
- [13] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [14] José Luis Garrido-Labrador. Procesamiento digital de imagen y video en Python con OpenCV. <https://github.com/jlgarridol/Python-OpenCV-Guide>, 2019.
- [15] Pieter Ginis, Alice Nieuwboer, Moran Dorfman, Alberto Ferrari, Eran Gazit, Colleen G Canning, Laura Rocchi, Lorenzo Chiari, Jeffrey M Hausdorff, and Anat Mirelman. Feasibility and effects of home-based smartphone-delivered automated feedback training for gait in people with parkinson's disease: a pilot randomized controlled trial. *Parkinsonism & related disorders*, 22:28–34, 2016.
- [16] Indeed. Salarios para empleos de a.c.s. informáticos en españa. <https://www.indeed.es/cmp/A.c.s.-Inform%C3%A1ticos/salaries>, jun 2019.
- [17] Alejandro R Jadad, R Andrew Moore, Dawn Carroll, Crispin Jenkinson, D John M Reynolds, David J Gavaghan, and Henry J McQuay. Assessing the quality of reports of randomized clinical trials: is blinding necessary? *Controlled clinical trials*, 17(1):1–12, 1996.
- [18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [19] Mario Juez-Gil. confluentinc/cp-docker-images. <https://github.com/confluentinc/cp-docker-images>.
- [20] Mario Juez-Gil. mjuez/spark-cluster-docker. <https://github.com/mjuez/spark-cluster-docker>.
- [21] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.
- [22] M Linares-del Rey, L Vela-Desojo, and R Cano-de la Cuerda. Aplicaciones móviles en la enfermedad de parkinson: una revisión sistemática. *Neurología*, 34(1):38–54, 2019.

- [23] MuyLinux. Red Hat lidera el segmento Linux en el mercado de servidores. <https://www.muylinux.com/2018/10/19/red-hat-lidera-mercado-linux-servidores/>.
- [24] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [25] William B Pennebaker and Joan L Mitchell. *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.
- [26] PkLab. How to make auto-adjustments(brightness and contrast) for image Android Opencv Image Correction. <https://answers.opencv.org/question/75510/how-to-make-auto-adjustmentsbrightness-and-contrast-for-image-android-opencv-image-correction/>, 2017.
- [27] Juan José Rodríguez Díez and Alvar Arnaiz González. Aprendizaje sobre flujos de datos, 1. introducción al análisis sobre flujos, apr 2020.
- [28] Greg Roelofs, Mark Adler, and Jean-loup Gailly. Zlib. <https://zlib.net/manual.html>, 2017.
- [29] Armin Ronacher, David Lord, Adrian Mönnich, and Markus Unterwaditzer. Welcome to jinja2. <http://jinja.pocoo.org/docs/2.10/>, 2008.
- [30] Armin Ronacher, David Lord, Adrian Mönnich, and Markus Unterwaditzer. Welcome to flask. <http://flask.pocoo.org/docs/1.0/>, 2010.
- [31] Gadiel Seroussi and Abraham Lempel. Lempel-ziv compression scheme with enhanced adapation, sep 1993. US Patent 5,243,341.
- [32] Wikipedia. Modelo de color hsv — wikipedia, la enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Modelo_de_color_HSV&oldid=118742027, 2019. [Internet; descargado 5-mayo-2020].
- [33] Wikipedia contributors. Bootstrap (front-end framework) — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Bootstrap_\(front-end_framework\)&oldid=895052706](https://en.wikipedia.org/w/index.php?title=Bootstrap_(front-end_framework)&oldid=895052706), 2019. [Online; accessed 16-May-2019].

- [34] Wikipedia contributors. JQuery — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=JQuery&oldid=896325154>, 2019. [Online; accessed 16-May-2019].
- [35] Wikipedia contributors. Distributed computing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Distributed_computing&oldid=956299584, 2020. [Online; accessed 20-May-2020].
- [36] Wensong Zhang et al. Linux virtual server for scalable network services.

MÁSTER UNIVERSITARIO EN
INTELIGENCIA DE NEGOCIO
Y BIG DATA EN
ENTORNOS SEGUROS



ANEXO

Manual de usuario



José Luis Garrido Labrador
José Miguel Ramírez Sanz

| Índice general

Índice general	I
Índice de figuras	II
1 Manual para el paciente	1
1.1. Controles	3
1.2. Llamadas	3
1.3. Evolución	3
2 Manual para el terapeuta	5
2.1. Inicio de sesión y modificación de contraseña	5
2.2. Llamadas	7
2.3. Evolución	9

| Índice de figuras

1.1. Botón de encendido del dispositivo.	1
1.2. Menú principal de la aplicación.	2
1.3. Mando de interacción con el sistema.	2
1.4. Imagen de ayuda de los controles.	3
1.5. Ejemplo pantalla de evolución.	4
2.1. Menú de inicio de sesión de la aplicación.	5
2.2. Menú principal de la aplicación.	6
2.3. Acceso a cambiar contraseña.	6
2.4. Menú de cambio de contraseña.	7
2.5. Menú de selección del paciente al que llamar.	8
2.6. Menú de la llamada con el paciente seleccionado.	8
2.7. Menú de selección del paciente en la evolución.	9
2.8. Menú de selección de acción en la evolución de un paciente.	9
2.9. Menú de insertar un medida con una fecha.	10
2.10. Lista de registros que se pueden modificar o eliminar.	10
2.11. Visualización de la evolución del paciente.	11

1 | Manual para el paciente

En este manual va a conseguir las herramientas necesarias para poder utilizar *Fis-Hubu* como herramienta rehabilitadora. Podrá encontrar en el mismo las instrucciones de acceso a las diferentes funcionalidades que ofrece la herramienta:

- **Llamar al médico:** en este modo podrá ponerse en contacto con su terapeuta para poder hacer su rehabilitación. Es importante que sepa que no siempre va a haber un médico al otro lado, por lo que es importante que concierte bien las citas con anterioridad.
- **Evolución:** en esta página podrá ver la evolución que ha tenido respecto a su última evaluación.
- **Ayuda:** en esta opción podrá acceder rápidamente a la ayuda para recordar cómo usar la aplicación y moverse por los menús.
- **Apagar:** de esta manera podrá apagar el dispositivo y ahorrar energía en su casa.

El primer paso que debe hacer es encender el dispositivo que le hemos instalado en casa. Para ello pulse el botón que se marca en la **Figura 1.1**. Nunca pulse ese botón una vez esté encendido o puede haber errores en la conexión.



Figura 1.1: Botón de encendido del dispositivo.

Una vez inicie el equipo verá una pantalla como la que se muestra en la [Figura 1.2](#). En el caso de ver el dibujo de un **dinosaurio** simplemente espere, el equipo se estará conectando.

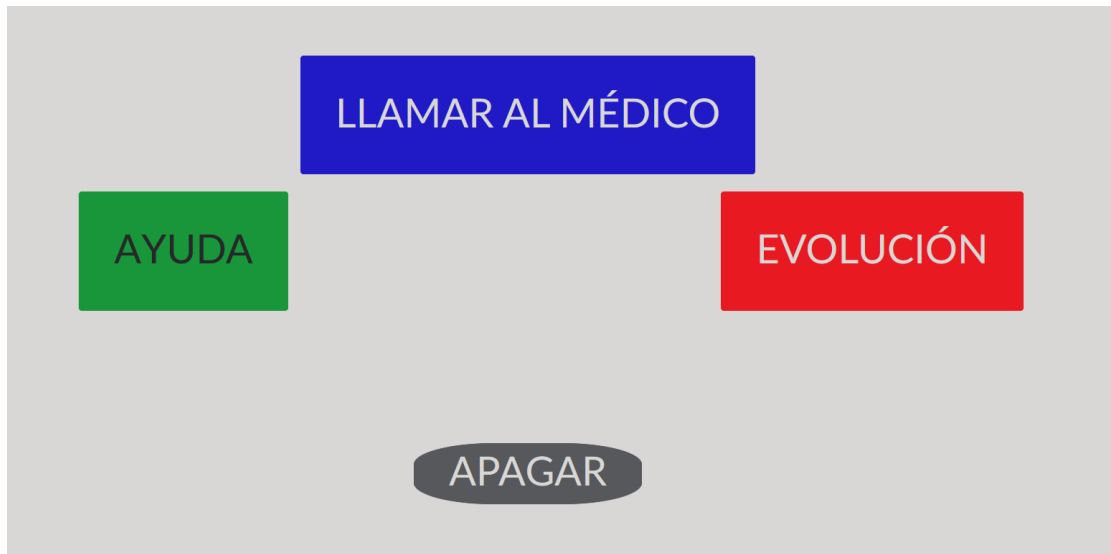


Figura 1.2: Menú principal de la aplicación.

Se le ha proporcionado un mando como el que se ve en la [Figura 1.3](#). Con él podrá acceder a las diferentes funciones que tiene la aplicación.



Figura 1.3: Mando de interacción con el sistema.

1.1. Controles

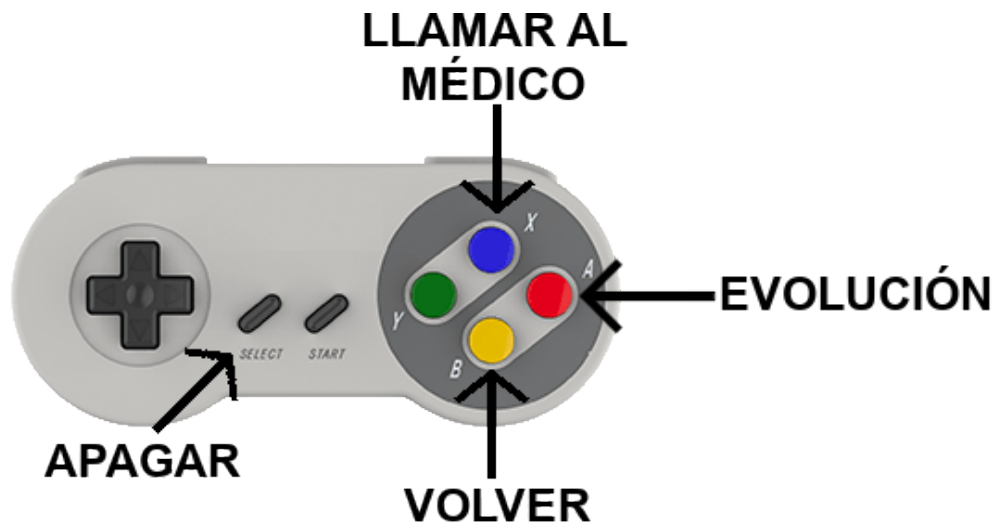


Figura 1.4: Imagen de ayuda de los controles.

En el mando que tiene hay cuatro botones con colores que representan los colores que verá en la pantalla. Con el botón azul podrá entrar en vídeo-llamada con el médico. Con el botón rojo ver su evolución. Con el botón verde ver la ayuda y con el botón amarillo volver a la pantalla principal.

Los botones de colores de su mando de televisión NO funcionan con la aplicación. Use únicamente el mando que le hemos proporcionado.

Para apagar el equipo pulse uno de los botones grises rectangulares. El equipo comenzará a apagarse, no lo desconecte ni pulse otro botón hasta que esté completamente apagado. Estará completamente apagado cuando el botón que ha pulsado para encenderlo se apague (deje de tener luz azul el botón de encendido).

Si no quiere esperar a que termine de apagarse, puede utilizar la televisión con normalidad. Es importante que no toque el equipo.

1.2. Llamadas

Cuando pulse el botón azul entrará en una vídeo-llamada. Si el médico no se ha conectado aún, únicamente verá su cámara. Cuando el médico esté le verá y le oirá con normalidad. Siga sus instrucciones para una mejor rehabilitación.

Para terminar la llamada, pulse el botón amarillo de su mando y volverá al menú principal.

1.3. Evolución

Pulsando el botón rojo podrá ver su evolución respecto a la última evaluación que ha recibido. Intente siempre mejorar, el terapeuta hará todo lo posible para que alcance una mejor calidad de vida.



Figura 1.5: Ejemplo pantalla de evolución.

2 | Manual para el terapeuta

En este manual va a encontrar la forma de ponerse en contacto con sus pacientes y conducirlos a una mejora de su calidad de vida a partir de la rehabilitación *online*. Lea atentamente estas instrucciones para poder ofrecer el mejor servicio a sus pacientes.

2.1. Inicio de sesión y modificación de contraseña

Para poder acceder a la aplicación le hemos ofrecido un identificador de usuario único y una contraseña. Introdúzcalos en los campos correspondientes como se observa en la [Figura 2.1](#).



Figura 2.1: Menú de inicio de sesión de la aplicación.

Una vez iniciada la sesión accederá al menú principal.



Figura 2.2: Menú principal de la aplicación.

Es recomendable que cambie su contraseña por una más segura, recuerde que trata con datos sensibles de pacientes, y la responsabilidad de la privacidad de esos datos es suya. Para cambiar la contraseña pulse el botón azul que se encuentra a la izquierda de su monitor. Como se observa en la [Figura 2.3](#) tendrá dos opciones, cambiar la contraseña o cerrar sesión. Si comparte equipo le recomendamos cerrar la sesión una vez finalice su trabajo.

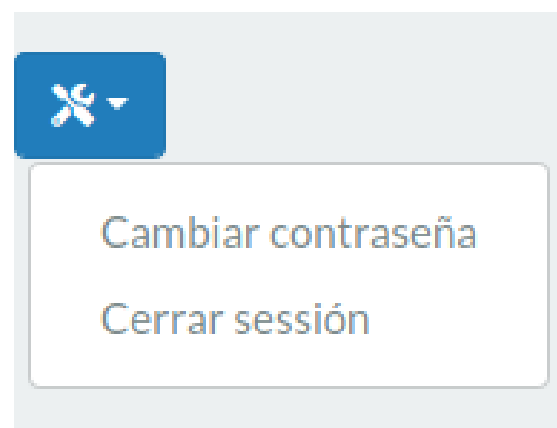


Figura 2.3: Acceso a cambiar contraseña.

Para cambiar la contraseña introduzca la que tiene actualmente y escriba la nueva dos veces en los dos campos como se observan en la [Figura 2.4](#).

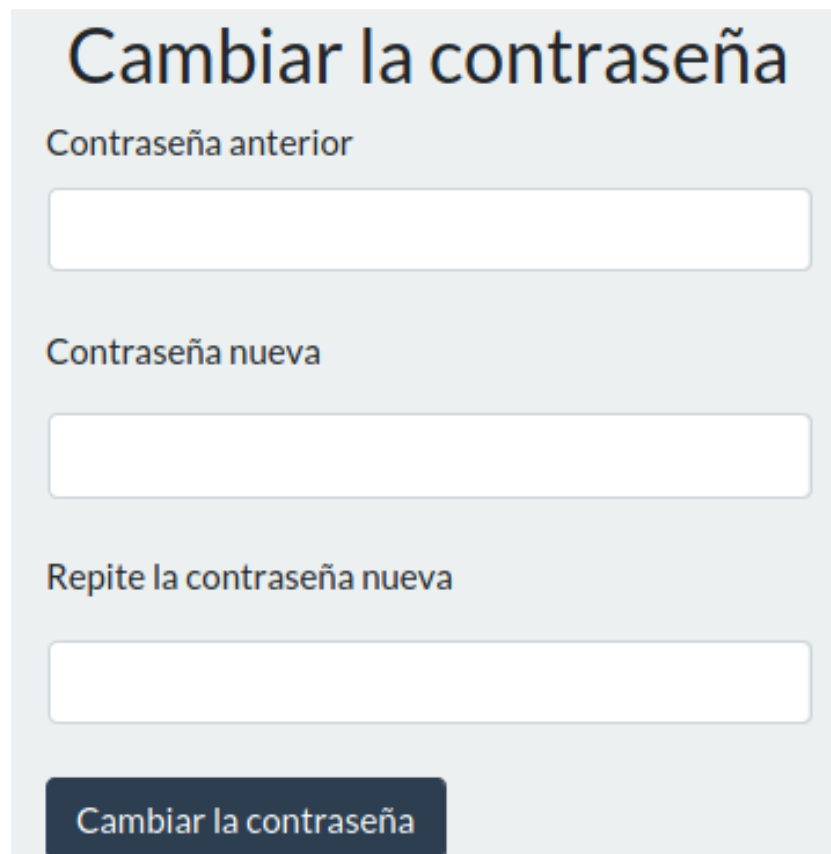
El formulario tiene un fondo gris claro. En la parte superior, el título "Cambiar la contraseña" está en un tamaño de fuente grande y en negrita. Debajo del título, hay tres campos de entrada de texto, cada uno con un label a su izquierda: "Contraseña anterior", "Contraseña nueva" y "Repite la contraseña nueva". Los campos de entrada son rectángulos blancos con una sombra suave. Al final del formulario, hay un botón rectangular de color azul oscuro con el texto "Cambiar la contraseña" en blanco.

Figura 2.4: Menú de cambio de contraseña.

Si perdiese la contraseña y no pudiese acceder al servicio, póngase en contacto con su responsable y haremos todo lo posible para ofrecerle una nueva.

2.2. Llamadas

Para comenzar la rehabilitación de sus pacientes, primero pulse **Comunicación** en el menú principal. Una vez dentro accederá a una pantalla como la que se observa en la [Figura 2.5](#). Seleccione el paciente con el que quiere comenzar la comunicación y pulse **Llamar**.

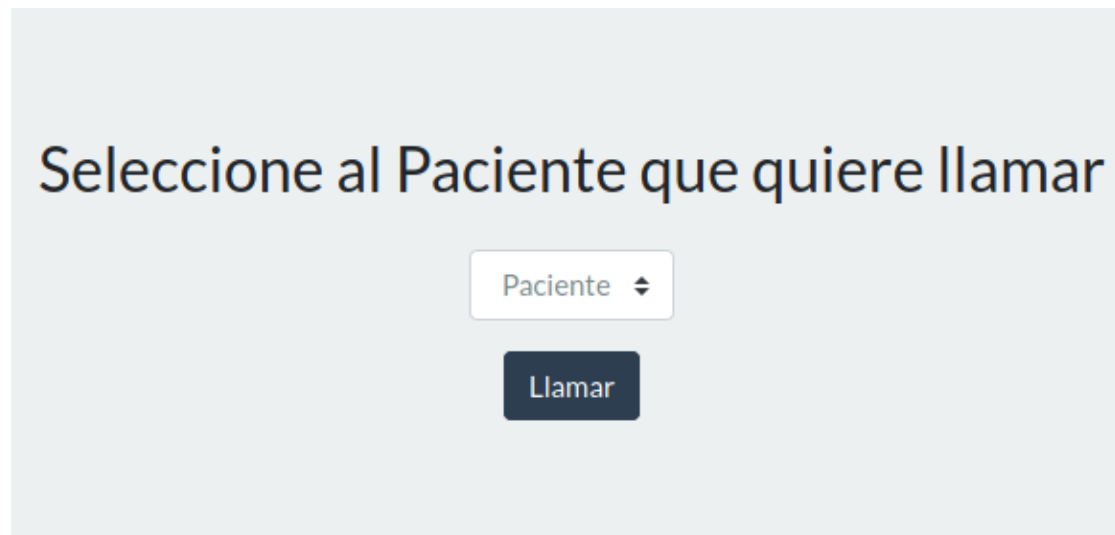


Figura 2.5: Menú de selección del paciente al que llamar.

Si el paciente no se ha conectado, no lo verá ni este recibirá ninguna notificación. Es importante que concertéis la cita con anterioridad. Igualmente usted tendrá el teléfono de contacto de cada uno de sus pacientes activos. Por favor, no llame a ningún paciente que pueda tener una reunión en ese momento y no es con usted.

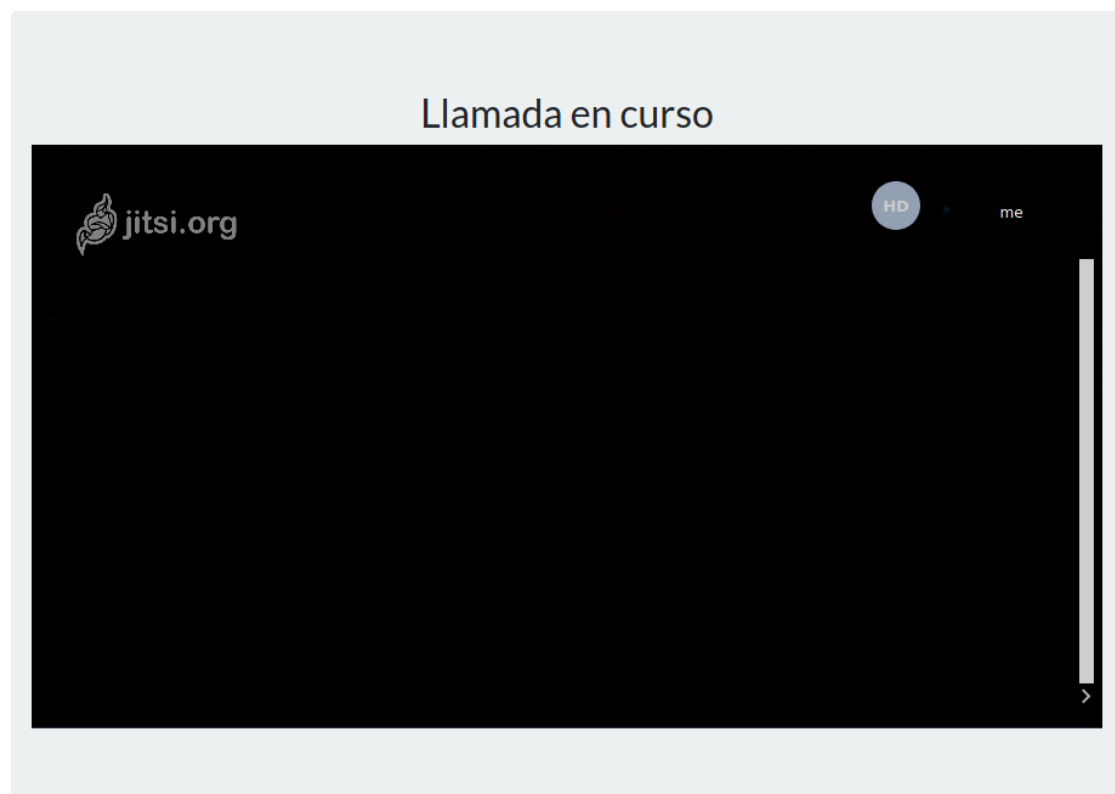


Figura 2.6: Menú de la llamada con el paciente seleccionado.

Para terminar la llamada pulse el botón «Volver hacia atrás» que habrá debajo de la misma.

2.3. Evolución

Para controlar la evolución que tiene su paciente tiene el menú «Evolución de pacientes». Cuando entre en el mismo, le aparecerá una pantalla como la [Figura 2.7](#) donde tendrá que elegir al paciente y pulsar «Menú de evolución»

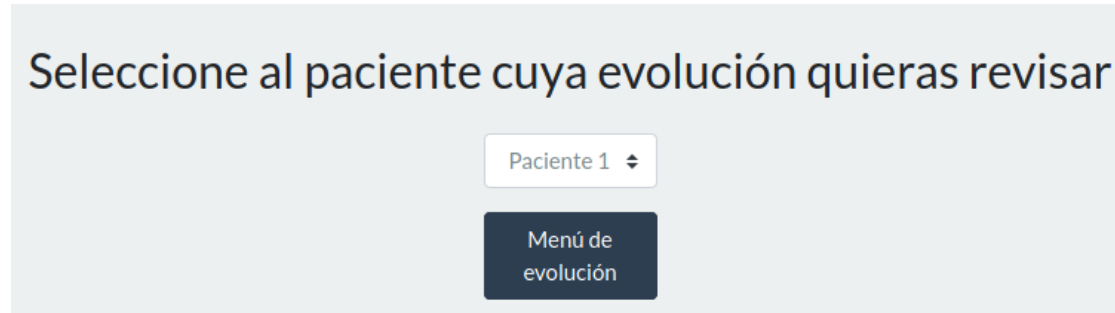


Figura 2.7: Menú de selección del paciente en la evolución.

Una vez dentro verá la opción de **añadir**, **modificar** o **ver** la evolución. El servicio que le ofrecemos incorpora un cálculo automatizado de las métricas de evolución, pero si ha tenido una rehabilitación fuera de este sistema podrá incorporarla. También podrá modificar y borrar las existentes y ver una gráfica de la evolución completa.

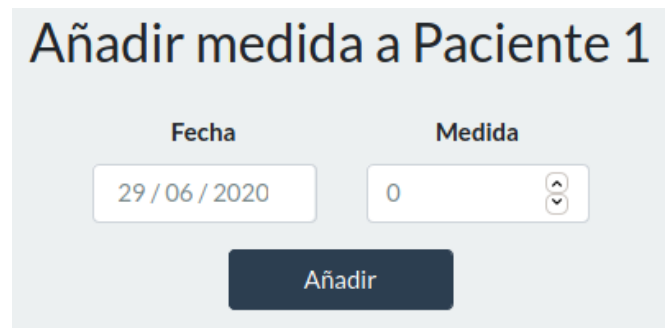


Figura 2.8: Menú de selección de acción en la evolución de un paciente.

Las medidas, cuando más altas sean indican una mejor calidad en los ejercicios realizados por los pacientes.

Creación y modificación

Podrá insertar nuevas medidas seleccionando su fecha e introduciendo su valor como se observa en la [Figura 2.9](#).



Formulario para añadir una medida a un paciente. El título es "Añadir medida a Paciente 1". Hay dos campos de entrada: "Fecha" con el valor "29/06/2020" y "Medida" con el valor "0". Debajo de estos campos hay un botón "Añadir".

Figura 2.9: Menú de insertar un medida con una fecha.

Para modificar o borrar una medida, en el menú de «Modificar medidas» encontrará la lista completa de medidas. Si cambia el valor de la medida y pulsa «Actualizar» el nuevo valor se guardará. Es importante que sepa que cualquier acto de actualización o borrado es permanente.



Fecha	Medida	Actualizar	Borrar
29-06-2020	15	Actualizar	Borrar
28-06-2020	1	Actualizar	Borrar
27-06-2020	10	Actualizar	Borrar

Figura 2.10: Lista de registros que se pueden modificar o eliminar.

Visualización

En cualquier momento, mientras el paciente tenga medidas, podrá ver la evolución que ha tenido en su historial de rehabilitaciones.



Figura 2.11: Visualización de la evolución del paciente.