UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior en Ingenierías Informática y de Telecomunicación

Máster Universitario en Ingeniería Informática

TRABAJO DE FIN DE MÁSTER

Sistema de Fog-Computing para Control Distribuido

Presentado por:
Víctor Vázquez Rodríguez

Tutor:
Antonio Javier Díaz Alonso
Departamento de Arquitectura y Tecnología de Computadores

Curso académico 2020-2021

Agradecimientos

Al Dr. Antonio Javier Díaz Alonso, tutor de este trabajo, por guiarme y aportarme su experiencia académica.

A mis padres, por darme la oportunidad de estudiar y seguir mi propio camino.

Al Dr. Juan Antonio Vázquez Rodríguez, mi hermano, por ser mi referente de esfuerzo, trabajo, bondad y humildad.

A mi novia, Rosa, por creer siempre en mí y estar a mi lado pase lo que pase.

A mis compañeros de promoción del Máster, por hacerme sentir como en casa lejos de ella.

Resumen

Con el advenimiento de la Industria 4.0, las empresas buscan incorporar técnicas de inteligencia artificial y análisis de datos a sus instalaciones y procesos industriales con el objetivo de mejorar la productividad y la autonomía. En este trabajo, se plantea la posibilidad de usar tecnologías de contenerización de procesos para el despliegue eficiente de estas nuevas tareas junto con las de tiempo real habituales en los sistemas de control industrial, estudiando las distintas tecnologías posibles y realizando un análisis del rendimiento de tareas de tiempo real contenerizadas con Docker. Además, se diseña e implementa una herramienta software que sirve como prueba de concepto para el despliegue y la orquestación de este tipo de tareas sobre entornos distribuidos mediante el uso de contenedores.

As Industry 4.0 gets closer, companies are looking to incorporate artificial intelligence and data analysis techniques to their facilities and industrial processes with the objective of improving productivity and autonomy. This paper proposes the use of processes containerization techonologies for efficient deployment of these new tasks along with the real-time tasks that are common in industrial control systems, studying different possible techonologies and analysing the performance of real-time tasks containerized using Docker. Finally, a proof-of-concept software tool for deployment and orchestration of this type of tasks on distributed environments by means of containers.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación y contexto

El Programma 101 es considerado por muchos como el primer ordenador de uso personal. Producido por la empresa italiana Olivetti entre los años 1962 y 1964, este dispositivos seasimilaba más a una calculadora que al concepto de ordenador que se tiene en la actualidad. Casi 60 años después, los ordenadores han evolucionado hasta convertirse en un objeto asequible y casi indispensable, teniendo la mayor parte de la sociedad acceso a algún dispositivo con capacidad de computación. Aspectos de nuestra vida como el ocio, serían muy diferentes sin las redes sociales o los servicios online. Esta evolución de los ordenadores y la computación en general continúa hoy en día, buscando conectar dispositivos cotidianos a internet e incorporar en ellos cierta capacidad de procesamiento. Televisores inteligentes en los que instalar aplicaciones, robots aspiradores que funcionan de manera autónoma, frigoríficos que permiten ver el estado de los alimentos que almacenan, todos estos productos surgen de la expansión de los ordenadores hasta todos los rincones de nuestras vidas, con el objetivo final de hacerla más sencilla para los humanos. Esta expansión no está ocurriendo solo en los hogares, si no que se avanza hacia un mundo más conectado donde aparentemente todo lo que nos rodea tenga acceso a la red.

La industria no es ajena a este cambio, ya que la aplicación del internet de las cosas (IoT por sus siglas en inglés) a los procesos industriales podría aportar beneficios muy importantes como son el aumento de la productividad, de la calidad de los productos o de la seguridad de las instalaciones. No obstante, se trata de un proceso de implantación complejo y de muy largo plazo, debido a los estrictos requisitos de algunos sectores industriales. Las fábricas y plantas de producción relegan las tareas de control de sus operaciones a los sistemas de control industriales o ICSs (*Industrial Control Systems*). Normalmente, estos sistemas deben responder ante eventos que ocurren en las instalaciones en ventanas de tiempo muy pequeñas, dependiendo del proceso concreto y sus riesgos asociados. La fiabilidad de estos sistemas y su tolerancia ante los fallos es de gran importancia, siendo crucial la verificación de estos aspectos durante su diseño y desarrollo. Es por estos requisitos tan estrictos que, en muchos casos, para la implantación de los ICSs se utilizan tanto *hardware* como *software* específicos y que ya han sido validados para este tipo de aplicaciones. No obstante, el uso de estas herramientas también plantea algunos inconvenientes, como por ejemplo la difícil interoperabilidad con otros sistemas debido a la naturaleza cerrada de las mismas o la reutilización del código en otras plataformas, lo que acorta la vida útil del *software*.

Desde esta situación se afronta el avance hacia la cuarta revolución industrial o Industria 4.0. Uno de los objetivos principales que se persigue es la automatización completa de los procesos industriales, haciando que sean independientes y autogestionados, aumentando así su eficiencia, productividad y seguridad. Para conseguir este objetivo, es necesario incorporar a estos procesos las nuevas técnicas de aprendizaje automático y análisis de grandes volúmenes de datos, construyendo sistemas experots capaces de tomar decisiones propias para su funcio-

namiento y gestión (p. ej., modificar el ritmo de producción en base a predicciones sobre la demanda). Aunque estos sistemas expertos se podrían desplegar en plataformas separadas de las que habitan los ICSs, esto duplica los costes de infraestructura y hace también más difícil el mantenimiento de la misma. Por ello, hay una tendencia cada vez mayor hacia la ejecución de tareas con distintos niveles de criticalidad en una misma plataforma. Por otra parte, las tareas de aprendizaje máquina suelen requerir de una capacidad de computación relativamente elevada, sobre todo cuando la cantidad de datos sobre la que se trabaja es grande. Un solo dispositivo no sería capaz de gestionar estas tareas de forma eficiente si, además, debe realizar tareas críticas de control, dificultando también que pueda cumplir con sus requisitos en ese aspecto. Aparecen entonces nuevos paradigmas de computación, como el *fog* o el *edge computing*, que pretenden decentralizar el procesamiento, alejándolo de la nube y llevándolo a elementos intermedios más cercanos a las fuentes de datos o a los dispositivos del borde de la red, respectivamente. En estos paradigmas, se persigue la máxima utilización de la capacidad de computación de los dispositivos presentes en la red, distribuyendo entre ellos las tareas necesarias de forma dinámica.

Dentro de este campo, se están llevando a cabo muchas investigaciones para validar la efectividad de estos nuevos modelos y hacerlos realidad, además de comprobar su efectividad en el ámbito industrial. Uno de los planteamientos más prometedores es el que conlleva la aplicación de tecnologías de virtualización a los sistemas de control industrial, permitiendo su despliegue y ejecución distribuida en convivencia con otros procesos. Estas tecnologías, que han sufrido un desarrollo masico en los últimos años debido al auge de la computación en la nube, pueden ahora ser claves para asegurar la resiliencia y robustez de los ICSs en entornos distribuidos.

Por todo esto, este trabajo abordará la complejidad de los problemas descritos intentando buscar mecanismos que simplifiquen la gestión de los sistemas *fog*/*edge*. Partiremos de la experiencia previa en trabajos con contenedores orientados al análisis de sus prestaciones para la ejecución de tareas con restricciones temporales, responsables de las ideas aquí propuestas y base fundamental de parte de la motivación de este trabajo. Además, esta línea de investigación tiene mucho potencial para mejorar los sistemas de control implementados en la actualidad, permitiendo por tanto una contribución importante tanto en aspectos industriales como científicos, lo que aumenta aún más nuestro interés en esta temática.

1.2. Objetivos

De cara a la realización de este trabajo, se han planteado los siguientes objetivos a cumplir:

- Comprender los conceptos básicos sobre sistemas operativos de tiempo real, algoritmos de planificación y sistemas de control industriales.
- Analizar las soluciones de tiempo real basadas en GNU/Linux existentes.
- Estudiar la viabilidad y rendimiento de las tecnologías de contenerización para la ejecución de tareas con restricciones temporales.
- Diseñar e implementar una herramienta de despliegue de tareas de tiempo real mediante contenedores que sirva como prueba de concepto.

- Caracterizar las prestaciones de la herramienta desarrollada, identificando posibles aplicaciones y/o mejoras de la misma.

Una parte relevante de los objetivos del presente proyecto están asociados al estudio y caracterización de tecnologías, así como al análisis del mercado, que se justifica por un enfoque innovador orientado al desarrollo de sistemas para *fog computing*. Esto, junto con los conocimientos adquiridos sobre implementación de sistemas de tiempo real, será clave para que la herramienta desarrollada como prueba de concepto sea lo más eficiente y funcional posible.

Por otra parte, en los objetivos también se hace hincapié en el uso de soluciones basadas en GNU/Linux, como parte de un fuerte compromiso con las herramientas libres y de código abierto. Algunos de los sistemas operativos de tiempo real más conocidos y usados son de código propietario y es necesario adquirir licencias para utilizarlos, lo cual no favorece ni el aprendizaje ni la reutilización del software desarrollado para estas plataformas. Últimamente, se han realizado contribuciones importantes al kernel de Linux para mejorar el soporte que ofrece a cargas de trabajo de este tipo, además de los múltiples proyectos más especializados que está llevando a cabo la comunidad. En esta aproximación, hemos considerado primordial la extensión del software libre, apoyando su uso siempre que sea posible. Por ello, la herramienta planteada se diseñará en torno al despliegue de tareas en sistemas GNU/Linux.

1.3. Planificación

Tradicionalmente, la planificación de las tareas en proyectos de desarrollo de software se realizaba siguiendo un modelo en cascada, donde antes de avanzar a la siguiente fase (p. ej., diseño, desarrollo, prueba) se debían completar todas las tareas de la anterior. Además, se intentaba estimar con precisión el tiempo necesario para desarrollar cada tarea, con el fin de obtener una predicción precisa de cúando estaría terminado el proyecto. Este tipo de metodología en cascada, en la que se planifica la totalidad del proyecto de antemano, es muy rígida y, por tanto, hace que sea más difícil adaptarse a los cambios que suceden durante el desarrollo del proyecto. En los últimos años ha cobrado fuerza una alternativa: las metodlogías ágiles. Lo que se plantea es afrontar la planificación y ejecución de las tareas de forma iterativa. En SCRUM, una de las metodologías ágiles más conocidas, se eligen tareas a realizar a corto plazo, lo que permite modificar las tareas a realizar si fuera necesario. En [1] se realiza una comparativa entre el modelo en cascada y las metodologías ágiles, llegando el autor a la conclusión de que estas últimas son siempre más eficaces para proyectos de pequeño y medio tamaño si se aplican de forma correcta. Es en los proyectos más complejos, donde hay múltiples equipos involucrados, en los que la aplicación de metodologías ágiles es más difícil y puede resultar ineficaz.

Observando esto y sabiendo que este proyecto es de baja complejidad, hemos decidido aplicar SCRUM para la planificación y la gestión de las tareas del mismo. Según este modelo, el desarrollo del proyecto se realiza en iteraciones, las cuales se planifican de forma dinámica y sobre la marcha. Al inicio del proyecto, se identifican las tareas que se consideran necesarias para la consecución de los objetivos, formando una pila o *backlog* de tareas. Los miembros del equipo estiman estas tareas en base a su experiencia. A diferencia del modelo tradicional, estas estimaciones no tienen que ser precisas, sino que lo que se busca es una medida orientativa del tamaño o complejidad de las tareas en relación con las demás. En muchos casos, para agilizar

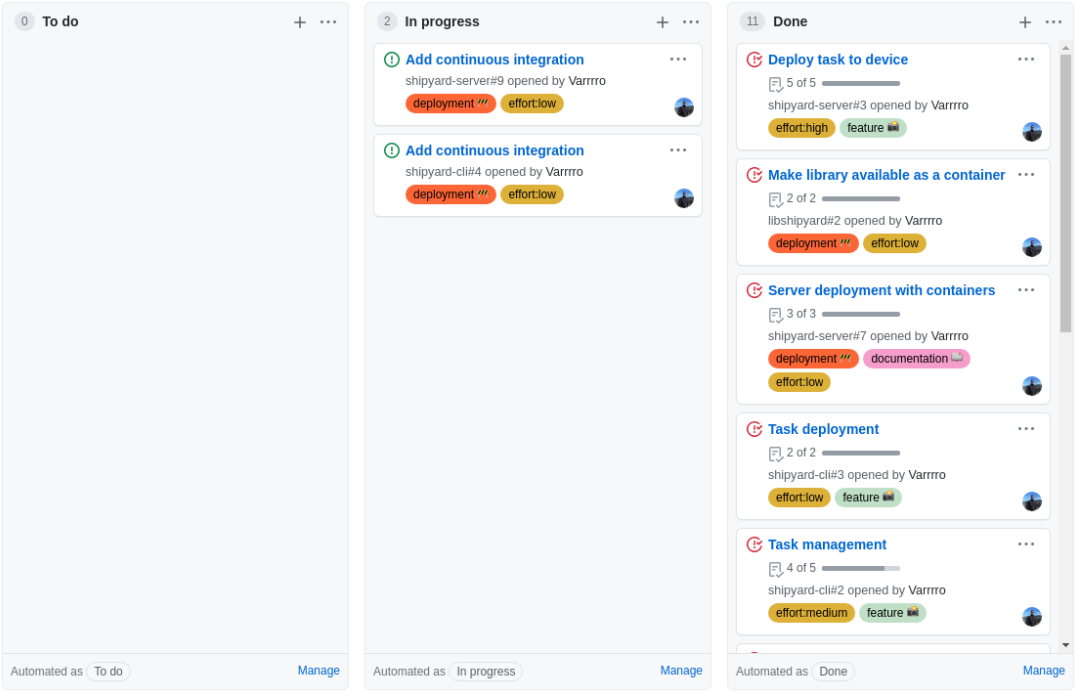

Figura 1.1.: Muestra del tablero Kanban usado para el seguimiento de las tareas del proyecto

el proceso de estimación, se aplican juegos. Una vez definida esta pila de tareas, comienzan las iteraciones o *sprints*, que suelen tener una duración pequeña, de 2 o 3 semanas. Antes de comenzar una nueva iteración, los miembros del equipo deciden qué tareas se van a acometer en la misma. La selección de estas tareas se puede basar en su estimación, su importancia o en el propio deseo de los miembros del equipo de realizarlas. Así, se realiza una planificación a corto plazo, lo que permite a los equipos tener mucho margen de reacción ante cambios en el proyecto (p. ej., cambios en los requisitos o problemas imprevistos). Una vez termina la iteración, el equipo evalúa el trabajo realizado, comprobando si las estimaciones que habían realizado al principio eran fidedignas, actualizando las de las tareas restantes si fuera necesario. Si se ha descubierto que se deben realizar más tareas de las inicialmente planteadas, éstas se pueden añadir a la pila entre iteraciones.

Como se puede apreciar, el modelo de SCRUM se centra mucho en el trabajo colaborativo dentro del equipo, por lo que el modelo que hemos aplicado en este proyecto se puede considerar como un SCRUM adaptado al trabajo individual. Las iteraciones seguidas han sido de dos semanas y se ha usado un tablero Kanban para el seguimiento de las tareas. En la figura 1.1 se muestra el tablero usado. El objetivo de estos tableros es ofrecer una visión clara del estado del proyecto con un simple vistazo, para lo que ordenan las tareas en tres columnas: Por hacer (*To do*), Haciendo (*In progress*) y Hecho (*Done*). A continuación, se listan las iteraciones realizadas con las tareas que se han acometido en cada una de ellas:

- **Iteración 1** (06/07 - 20/07)
  - (Documentación) Estudiar el funcionamiento de los RTOS.

  - (Documentación) Estudiar las soluciones de tiempo real para Linux.
  - (Documentación) Estudiar las técnicas de diseño e implementación de tareas de control.
  - (Documentación) Estudiar el uso de contenedores para tareas de tiempo real.

- **Iteración 2** (20/07 - 03/08)
  - (Servidor) Implementar la obtención, inserción y eliminación de nodos.
  - (Servidor) Implementar la obtención, inserción y eliminación de tareas.

- **Iteración 3** (03/08 - 17/08)
  - (Servidor) Implementar el despliegue y la eliminación de tareas en los nodos.

- **Iteración 4** (17/08 - 31/08)
  - (Servidor) Implementar la obtención, inserción y eliminación de tareas.
  - (Servidor) Implementar el despliegue y la eliminación de tareas en los nodos.
  - (Servidor) Implementar la actualización de nodos y tareas.
  - (Librería) Implementar las funciones de manipulación de los atributos de planificación.
  - (Librería) Crear imagen base.

- **Iteración 5** (31/08 - 14/09)
  - (Servidor) Mejorar el manejo de los errores.
  - (Servidor) Definir despliegue mediante contenedores.
  - (Cliente) Implementar la gestión de los nodos.
  - (Cliente) Implementar la gestión de las tareas.
  - (Cliente) Implementar el despliegue de tareas.

- **Iteración 6** (14/09 - 28/09)
  - (Servidor) Implementar el despliegue y la eliminación de tareas en los nodos.
  - (Cliente) Implementar la gestión de los nodos.
  - (Cliente) Implementar el despliegue de tareas.

  - (Librería) Implementar las funciones de manipulación de los atributos de planificación.
  - (Librería) Crear imagen base.

- **Iteración 7** (28/09 - 12/10)
  - (Experimentación) Caracterizar el impacto de los contenedores sobre el rendimiento en tareas de tiempo real.

- **Iteración 8** (12/10 - 26/10)
  - (Experimentación) Caracterizar el impacto de los contenedores sobre el rendimiento en tareas de tiempo real.

- **Iteración 9** (26/10 - 09/11)
  - (Experimentación) Caracterizar el rendimiento del sistema desarrollado.

- **Iteración 10** (09/11 - 23/11)
  - (Servidor) Definir flujo de integración continua.
  - (Cliente) Definir flujo de integración continua.
  - (Librería) Definir flujo de integración continua.

Como se puede apreciar, las tareas se han diferenciado según su tipo o el componente del sistema desarrollado al que se refieren. Además, para algunas de estas tareas ha sido necesario dedicar más de una iteración. En el caso de la tarea de implementación de las operaciones de obtención, inserción y eliminación de tareas, se tuvo que volver a incorporar en la iteración 4 debido a algunos errores cometidos en la implementación y que se detectaron más tarde. Otro caso similar fue la implementación de la funcionalidad de despliegue de tareas en los nodos. En el código inicialmente escrito en las iteraciones 3 y 4, esta operación se llevaba a cabo abriendo directamente una conexión SSH con el nodo objetivo y ejecutando los comandos de despliegue, emulando lo que haría un operario humano. Posteriormente, se llegó a la conclusión de que esta implementación se podía mejorar la usar la librería oficial de Docker para Python, con la cuál se podía establecer una conexión al motor Docker del nodo remoto para realizar el despliegue. Estas situaciones, junto con otras dadas en más tareas, reflejan fielmente la capacidad de adaptarse a los imprevistos que ofrece SCRUM que ya hemos comentado anteriormente.

1.4. Material y métodos

En esta sección, se presentan todas las herramientas que se han usado para llevar a cabo este proyecto, además de las metodologías de trabajo seguidas. En la figura 1.2 se puede ver una recopilación de los logotipos de todas estas herramientas, las cuales se van a exponer en detalle a continuación.

En un proyecto de desarrollo de código como es este, una de las herramientas más importantes es la de control de versiones. En nuestro caso, se ha usado **Git** debido a que es una herramienta de código libre y, además, es la más conocida y usada. El repositorio remoto se ha alojado en **GitHub**, la popular plataforma de desarrollo de código colaborativo. El desarrollo de código como tal se ha hecho con **Visual Studio Code**, un editor de texto de Microsoft que, aunque ofrece menos funcionalidad de serie que los entornos de desarrollo (IDEs) especializados, también es mucho más liviano y rápido. Además, esta funcionalidad se puede extender enormemente mediante extensiones, pudiendo convertirlo casi en un IDE adaptado a las necesidades concretas del usuario. Gracias a estas extensiones, se puede usar el mismo editor para distintos programar en distintos lenguajes, lo cual ha sido una de las principales razones por las que se ha escogido.

Figura 1.2.: Logotipos de las herramientas utilizadas. De izquierda a derecha yde arriba a abajo: Python, Docker, Raspberry Pi, GitHub, VS Code y draw.io

Como ya se ha indicado en la sección anterior, se ha aplicado un SCRUM adaptado como metodología de planificación, gestión y seguimiento del proyecto, usando un tablero Kanban para la visualización de las tareas. Concretamente, se ha usado el tablero que ofrece GitHub como parte de sus herramientas para la gestión de proyectos. Las tareas se añaden a los repositorios como *issues*, los cuales se añaden al tablero para controlar su estado de realización. El beneficio que nos aporta el uso de GitHub para esto es el de tener una plataforma unificada tanto para la gestión del proyecto como para su desarrollo. Esta integración permite flujos de trabajo muy interesantes, como por ejemplo referenciar los *issues* ya mencionados en los mensajes de *commit*, de forma que se vinculan los cambios realizados con la tarea en la que se engloban. La gestión del proyecto es, por tanto, más directa y sencilla.

Además del tablero de proyecto y los propios repositorios de Git, también se ha hecho uso de **Actions**, la herramienta de CI/CD de GitHub. Como su propio nombre indica, se definen acciones en flujos de trabajo que se ejecutan para los eventos del repositorio (p. ej., un nuevo *push*, la publicación de una nueva versión del producto, un *pull request*) que desee el desarrollador. Estas acciones pueden incluir, por ejemplo, la ejecución de pruebas, la construcción de artefactos o el despliegue del software sobre diversas plataformas. Así, se consigue automatizar los procesos de despliegue y se incorporan también medidas para el control de la calidad del código. En nuestro caso, en la ejecución de las pruebas se genera un informe sobre la cobertura

del código, el cuál se envía a **Codecov**, una plataforma especializada en este tipo de informes, para su análisis y visualización.

Los diagramas que se presentan a lo largo de este trabajo, incluidos los que componen el diseño de la herramienta desarrollada y sus componentes, se han creado con la herramienta online **draw.io**. Al integrarse directamente con Google Drive, se puede usar desde el navegador, siendo una solución muy cómoda y accesible para la creación de estos diagramas.

En cuanto al desarrollo y despliegue de la herramienta planteada, se han utilizado varias tecnologías. Los lenguajes de programación son Python y C, muy conocidos y utilizados en la industria. C es el lenguaje de referencia para la programación de sistemas y las tareas de bajo nivel, mientras que Python ofrece una sintaxis muy sencilla para la implementación de herramientas como aplicaciones o servidores. Python también posee una nutrida colección de librerías para distintos casos de uso, lo cual también fue muy atractivo para su elección. En este proyecto, se usan varias de estas librerías para facilitar la implementación de algunos aspectos de la herramienta. Destacamos las siguientes:

- **hug** - Escritura de APIs sencilla y sin ataduras. Esta librería trata de competir con otras más conocidas como Flask, proponiendo un modelo mucho más simple y dando más libertad al desarrollador.

- **click** - Más que una librería, se trata de un *framework* para la implementación de aplicaciones de línea de comandos (CLI). Ofrece todas las utilidades que se pueden necesitar para este tipo de aplicaciones, como es la generación automática de páginas de ayuda.

- **marshmallow** - Esta librería permite la sencilla deserialización y serialización de objetos JSON a objetos Python. Esto es especialmente útil al trabajar con servicios web, ya que se pueden leer los datos JSON recibidos y validarlos frente a un esquema de datos definido por el desarrollador.

- **pymongo** - Ejecución de consultas sobre bases de datos MongoDB, que son las usadas en este proyecto.

- **paramiko** - Herramientas para trabajar con SSH desde Python. Esencial para el despliegue de las tareas sobre los nodos.

- **docker** - La librería oficial de Docker para Python. Con ella, se puede conectar con un servidor Docker (local o remoto) para ejecutar operaciones de construcción de imágenes o lanzamiento de contenedores.

- **requests** - Es la librería más conocida para la realización de peticiones HTTP en Python.

- **unittest** - Aunque no se trata de una librería externa como las demás, ya que forma parte de la librería estándar de Python, merece ser destacada por ser la utilizada para la escritura y ejecución de las pruebas del código desarrollado.

Además de estas librerías, se han usado otras para el análisis y formateo del código como son **pylint** y **autopep8**.

Figura 1.3.: Raspberry Pi 4B utilizada en el trabajo

Como ya se ha mencionado, la herramienta desarrollada trabaja con **MongoDB** para la gestión del almacenamiento persistente de los datos y el acceso a los mismos. Se trata de un gestor de bases de datos documental que pertenece a la familia NoSQL (*Not only SQL*). Las bases de datos documentales divergen del modelo relacional tradicional que siguen otros gestores tan famosos como MySQL o PostgreSQL. No existen los conceptos de tabla o fila, sino que se almacenan colecciones de documentos no estructurados. En el caso de MongoDB, se trabaja con documentos JSON, formato ampliamente usado en el entorno web para la transmisión de datos debido a su legibilidad y velocidad de serialización/deserialización. Cuando decimos que los documentos son no estructurados, nos referimos a que no tienen por qué poseer los mismos campos, ya que no hay un esquema definido a seguir. MongoDB no cumple con ACID, pero es una pérdida aceptable a cambio de la flexibilidad y facilidad de uso que nos ofrece. Además, el lenguaje de consultas es mucho más claro e intuitivo que SQL, permitiendo realizar consultas de envergadura similar.

La idea principal en torno a la cual gira este trabajo es la del uso de contenedores para el despliegue de tareas de tiempo real. En este sentido, **Docker** es la tecnología de contenerización en la que nos hemos apoyado para la implementación de la herramienta propuesta. Esta elección recae principalmente en el hecho de que se trata del motor de contenedores más conocido y utilizado. De la mano de Docker, también se ha utlizado **Docker Compose** para el despliegue sencillo de un entorno de desarrollo completo que permita replicar un despliegue de producción de forma simplificada. Las imágenes de contenedores definidas en este proyecto, se han alojado en el repositorio de imágenes **DockerHub**, aprovechando que proporciona un sistema de construcción automática de imágenes.

Para realizar las pruebas del sistema desarrollado, se ha usado una **Raspberry Pi 4B**, con-

cretamente la que se muestra en la figura 1.3. Se trata de un SBC (*Single Board Computer*) de bajo coste y muy versátil, pudiendo usarse para infinidad de aplicaciones. Las características detalladas de esta plataforma se pueden ver en la tabla 1.1.

Procesador	Cortex-A72
Arquitectura	ARMv7
Número de núcleos	4
Frecuencia de reloj	1500 MHz
RAM	4 GB
Sistema operativo	Raspberry Pi OS Lite 4.19

Tabla 1.1.: Especificaciones de la Raspberry Pi usada en el trabajo

El sistema operativo de este dispositivo es GNU/Linux (concretamente, basado en Debian), y se le ha aplicado la revisión 24 del parche de tiempo real **PREEMPT-RT**. Este parche aporta al sistema la capacidad de realizar planificación de procesos apropiativa.

1.5. Estructura de la memoria

En el capítulo 1, se ha introducido el proyecto y explicado tanto la motivación detrás del mismo como los objetivos planteados, la planificación seguida y las herramientas, tecnologías y metodologías que se han utilizado.

En el capítulo 2, se realiza un estudio en profundidad del estado de la técnica en lo relativo a los sistemas confiables, la computación en la nube y sus derivados, las tareas de tiempo real y las tecnologías de virtualización. Se introducen todos los conceptos relevantes, además de presentar otros trabajos interesantes llevados a cabo en estas áreas.

En el capítulo 3, se analiza el rendimiento de los procesos de tiempo real contenerizados, haciendo especial hincapié en el impacto que tienen sobre el mismo las tecnologías de contenerización frente a procesos no virtualizados.

En el capítulo 4, se aborda el diseño de una herramienta para el despliegue y seguimiento de tareas de tiempo real en entornos distribuidos, así como su implementación y prueba. Se presentan los resultados obtenidos y se discute sobre los mismos.

En el capítulo 5, se realiza una revisión del proyecto completo y se presentan las conclusiones, así como las maneras en las que se puede extender y mejorar el trabajo realizado.

2. Revisión del estado de la técnica

Antes de poder aventurarnos el desarrollo de una herramienta para la orquestación de tareas de tiempo real usando contenedores, es necesario estudiar y comprender los fundamentos y peculiaridades de diversos campos como son los sistemas de tiempo real, la planificación de procesos, la computación en la nube o las tecnologías de virtualización. En este capítulo, se introducen los conceptos más importantes de estos campos, además de revisar otros trabajos realizados con el objetivo de comprobar cuál es el estado de la investigación en dichos temas.

2.1. Sistemas empotrados, confiables y de criticalidad mixta

Los ordenadores y teléfonos inteligentes que usamos a diario nos permiten realizar multitud de tareas diferentes: desde leer el correo o navegar por internet hasta editar imágenes y vídeos o realizar videollamadas. Aunque no nos demos cuenta, interactuamos habitualmente con muchos más sistemas informáticos además de los ya mencionados. La computación está presente en los coches, los trenes, los aviones, los satélites espaciales, los televisores o las lavadoras. Estos sistemas, que reciben el nombre de sistemas empotrados, son diseñados para llevar a cabo de manera óptima un conjunto de tareas específico, frente al enfoque generalista de los ordenadores de uso personal. Como se introduce en [2], los sistemas empotrados son comunes en contextos en los que el rendimiento es primordial, como son las comunicaciones de red o la compresión/decompresión de audio y vídeo para retransmisiones en directo. En este libro se exponen diversas aplicaciones de los sistemas empotrados de alto rendimiento para sistemas ciber-físicos o CPS (*Ciber-Physical Systems*). Los CPS son, en esencia, sistemas informáticos que interactúan con procesos físicos, actuando en función de los cambios en su entorno. Este aspecto hace que el diseño y la implementación de los CPS difiera considerablemente del resto de sistemas empotrados, ya que hay que prestar especial atención a las características del entorno en el que se desplegará el sistema y a los mecanismos de entrada y salida (interacción con el entorno), además de optimizar el consumo de memoria y procesamiento para cumplir con las limitaciones del hardware [3].

En algunos casos, estos sistemas son críticos, lo que significa que un fallo en su funcionamiento supone daños graves a las personas o al medio ambiente. Este es el caso de los aviones, los trenes o las plantas nucleares, por ejemplo. En estos sistemas, cobra especial importancia el concepto de confiabilidad, es decir, garantizar el correcto funcionamiento del sistema en todo momento. A nivel de software, existen arquitecturas y patrones de diseño orientados a garantizar la tolerancia ante los fallos del mismo [4][5]. Por otra parte, la replicación de componentes [6] es una técnica muy usada tanto para el software como el hardware. Lo que se intenta con la replicación es asegurar que un cálculo o procesamiento se lleva a cabo de forma correcta, aunque alguna de las réplicas falle. En [7], se realiza una revisión de los sistemas de control tolerantes ante fallos dividiéndolos en tres tipos: AFTCS (*Active Fault Tolerant Control Systems*), PFTCS (*Passive Fault Tolerant Control Systems*) o HFTCS (*Hybrid Fault Tolerant Control*

Systems). Para cada uno de estos tipos, los autores presentan las principales arquitecturas usadas, los modelos de análisis matemático usados para su validación y las últimas técnicas usadas para su diseño. Por otra parte, el estudio realizado en [8] tiene como objetivo analizar la aplicación de los estándares de seguridad, protección y privacidad al diseño y desarrollo de sistemas confiables, llegando los autores a la conclusión de que cada vez están ganando más popularidad los de protección y privacidad, aunque los procesos para asegurar estos aspectos son menos maduros que los relacionados con la seguridad. Además, también se identifica una falta de acción combinada en estos aspectos, trabajando normalmente por separado en cada uno de ellos.

Para muchos sistemas de control, el tiempo es también un aspecto relevante a la hora de garantizar su correcto funcionamiento. Estos son los llamados sistemas de tiempo real. En un sistema normal, en el que se obtiene una salida al aplicar una cierta lógica sobre la entrada, la corrección de dicha salida depende de la corrección de la lógica aplicada, es decir, del cálculo realizado. No obstante, cuando hablamos de sistemas de tiempo real, la corrección de la lógica no es suficiente para determinar que el funcionamiento es correcto, sino que esta corrección depende también del momento temporal en el que se obtiene [9]. En otras palabras, si el cálculo es correcto pero el resultado no llega en el momento en el que se necesita, se considera como un fallo del sistema. Las tareas que ejecuta un sistema de tiempo real están, por tanto, sujetas a restricciones temporales. Normalmente, hablamos de un tiempo límite o *deadline* antes del cuál se debe haber completado la tarea. Comúnmente, se suelen clasificar las tareas en dos tipos dependiendo de las consecuencias de incumplir con sus restricciones temporales:

- Tareas de tiempo real «blandas»: Se denominan así aquellas tareas en las que no completar las mismas antes del límite supone una reducción en la calidad del servicio (QoS).
- Tareas de tiempo real «duras»: En estas tareas, el no cumplir con las restricciones temporales supone un fallo grave del sistema con posibles consecuencias catastróficas.

Para implementar estos sistemas, se hace uso de herramientas y algoritmos de planificación de procesos específicos que permiten la ejecución de estos tipos de tareas cumpliendo con sus restricciones temporales. Esto se explica en más detalle en la sección 2.4.

En este trabajo, nos centraremos especialmente en los sistemas de control aplicados a entornos industriales. Debemos diferenciar entre dos conceptos: PCS (*Process Control System*), que es el sistema encargado de controlar una parte concreta de la planta (p. ej., una turbina o un brazo robótico), e ICS (*Industrial Control System*), que se refiere al control de toda la planta al completo. Un ICS está compuesto, por tanto, de múltiples PCS que se encargan de controlar los distintos aspectos del proceso de producción. Estos PCSs son sistemas empotrados como los que hemos comentado a los que se aplican también los conceptos de confiabilidad y seguridad. En [10] se realiza un estudio del estado del arte en cuanto a la seguridad de los ICS, concluyendo que, aunque las vulnerabilidades de muchos de los sistemas actuales son conocidas, la aplicación de parches que las solucionen son inviables debido a los altos costes asociados a la recertificación o directamente debido a la incompatibilidad con algunos sistemas más antiguos. Es necesario, por tanto, incorporar los conceptos de seguridad a los procesos de diseño desde el primer momento para evitar dar lugar a estas situaciones.

En 2007, Vestal [11] publica una primera propuesta para la planificación de conjuntos de tareas de criticalidad mixta. Este importante avance es considerado por muchos como el inicio

de la investigación en sistemas de criticalidad mixta o MCS (*Mixed-Criticality Systems*). La idea detrás de estos sistemas es, como su propio nombre indica, poder ejecutar sobre una misma plataforma hardware tanto tareas críticas como tareas no críticas o con niveles de criticalidad menores, asegurando en todo momento que las restricciones temporales se cumplen, al menos para las tareas más críticas. Desde entonces, se han planteado muchos modelos para la implementación de estos sistemas. En [12], Burns realiza una revisión de toda la investigación realizada en este campo hasta marzo de 2019. Se muestran en esta revisión algunas arquitecturas propuestas, además de las principales técnicas de análisis para estos sistemas en plataformas uniprocesador y multiprocesador. Burns identifica la conciliación entre la separación de los procesos y la compartición de los recursos como el principal problema de los MCS. En este aspecto, nuestro trabajo propone el uso de contenedores como medio de ejecución de los distintos procesos sobre una misma plataforma consiguiendo esa separación.

No podemos terminar nuestra revisión de los sistemas empotrados sin hablar sobre el uso del kernel de Linux para su implementación. Como referencia, hemos tomado dos estudios realizados en 2004 sobre el uso de Linux para sistemas empotrados [13][14]. En el primero, se destaca la buena situación de Linux en este campo, suponiendo las soluciones comerciales basadas en el kernel de Linus Torvalds el 15,5 % del mercado. El segundo estudio es una encuesta realizada a 268 personas que trabajan en el campo de los sistemas empotrados, ya sea académicamente o de forma comercial. Lo más destacable es que la mayoría de los participantes usan Linux en sistemas empotrados para comunicaciones, dispositivos móviles o control de maquinaria. Esto último es especialmente alentador para este trabajo, en el que pretendemos usar Linux como base para la implementación de ICS.

2.2. La nube en los sistemas industriales: fog/edge computing para sistemas de tiempo real

2.3. Tecnologías de virtualización: hipervisores y contenedores

2.4. Aspectos de computación en tiempo real: RTOS, algoritmos de planificación y herramientas

A. Estimación de costes del proyecto

Al tratarse, mayoritariamente, de un proyecto de Ingeniería del Software, se ha realizado también una estimación de los costes asociados a la realización del mismo. Para ello, se ha aplicado un modelo basado en tiempo y materiales. En 2019, el coste salarial medio de un ingeniero informático recién salido de la universidad en España era de entre 24.000 y 28.500 euros brutos al año. En esta estimación, se ha asumido un salario mensual de 2.000€ brutos al mes, que es aproximadamente un salario anual de 24.000€. Las tareas de coordinación y dirección del jefe del proyecto se estiman en un 10 % del trabajo de ingeniería, con un coste medio mensual de unos 5.000€ al mes para un ingeniero sénior. Además, se ha supuesto también una jornada laboral que llega al máximo en España de 40 horas semanales. También se tiene en cuenta el coste del hardware usado para pruebas. La estimación final del coste de desarrollo de este proyecto se muestra en la tabla A.1.

Raspberry Pi 4B 4GB	60,00€
Cable Ethernet	7,00€
Cable de alimentación USB-C	7,00€
Mano de obra ingeniero	8.000,00€
Mano de obra ingeniero jefe	20.000,00€
	28.074,00€

Tabla A.1.: Desglose de costes del proyecto

A todo esto habría que sumarle el coste de la estación de trabajo usada para el desarrollo del proyecto, la cuál consiste de un ordenador de sobremesa o portátil y los periféricos necesarios, junto con los gastos asociados al consumo eléctrico y el acceso a internet. Estos gastos se han omitido de la estimación realizada debido a que son muy variables y tampoco tienen un impacto muy representativo en los costes del proyecto.

Bibliografía

[1] W. V. Casteren, "The Waterfall Model and the Agile Methodologies : A comparison by project characteristics," 2017. Publisher: Unpublished.

[2] M. Wolf, *High-Performance Embedded Computing: Applications in Cyber-Physical Systems and Mobile Computing*. Elsevier, 2 ed., Apr. 2014.

[3] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. MIT Press, 2 ed., Dec. 2016. Google-Books-ID: chPiDQAAQBAJ.

[4] L. L. Pullum, *Software Fault Tolerance Techniques and Implementation*. Artech House, 2001. Google-Books-ID: O50vDwAAQBAJ.

[5] B. Randell, "System structure for software fault tolerance," *ACM SIGPLAN Notices*, vol. 10, pp. 437–449, June 1975.

[6] A. Schiper, "Dependable Systems," in *Dependable Systems: Software, Computing, Networks* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. Kohlas, B. Meyer, and A. Schiper, eds.), vol. 4028, pp. 34–54, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[7] A. A. Amin and K. M. Hasan, "A review of Fault Tolerant Control Systems: Advancements and applications," *Measurement*, vol. 143, pp. 58–68, Sept. 2019.

[8] L. Shan, B. Sangchoolie, P. Folkesson, J. Vinter, E. Schoitsch, and C. Loiseaux, "A Survey on the Application of Safety, Security, and Privacy Standards for Dependable Systems," in *2019 15th European Dependable Computing Conference (EDCC)*, (Naples, Italy), pp. 71–72, IEEE, Sept. 2019.

[9] A. Gambier, "Real-time control systems: a tutorial," in *Proceedings of the 5th Asian Control Conference (IEEE Cat. No.04EX904)*, vol. 2, (Melbourne, Australia), pp. 1024–1031, IEEE, July 2004.

[10] M. Krotofil and D. Gollmann, "Industrial control systems security: What is happening?," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 670–675, July 2013. ISSN: 2378-363X.

[11] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, (Tucson, AZ, USA), pp. 239–243, IEEE, Dec. 2007.

[12] A. Burns and R. I. Davis, "Mixed Criticality Systems - A Review," 2015.

[13] D. Geer, "Survey: Embedded Linux Ahead of the Pack," *IEEE Distributed Systems Online*, vol. 5, pp. 3–3, Oct. 2004.

[14] J. Henkel and M. Tins, "Munich/MIT Survey: Development of Embedded Linux," Jan. 2004.