Trabajo Fin de Grado en Ingeniería Informática

Infraestructura software para el arbitraje multiestratégico sobre criptomonedas

Daniel Huici Meseguer

Director: Ricardo J. Rodríguez

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Junio de 2021
Curso 2020/2021

Agradecimientos

A Ricardo, por guiarme y brindarme esta oportunidad.
A mi familia, por confiar en mí.
A mis amigos, por acompañarme en este camino.

RESUMEN

En la economía y las finanzas, el arbitraje supone una estrategia que permite sacar provecho de las diferencias de precio entre varios mercados financieros sin que apenas suponga un riesgo. Esta estrategia puede verse potenciada en un mercado moderno como el de las criptomonedas, dado que es un tipo de moneda completamente descentralizada donde la criptografía juega un papel fundamental para asegurar transacciones y donde, además, existen diferencias acentuadas entre los mercados debido a su alta volatilidad.

Debido a que los valores de los mercados fluctúan cada muy poco tiempo y que existe una de gran cantidad de mercados, tratar de encontrar oportunidades de arbitraje de forma manual resulta una tarea muy complicada. Por ello, en este TFG se ha diseñado un sistema basado en la tecnología Elixir que monitoriza de forma automatizada varias plataformas de cambio de criptomonedas, pudiendo compararlos y obtener posibles beneficios, permitiendo varias estrategias de arbitraje distintas.

El diseño arquitectural del sistema permite la inclusión de nuevos mercados, estrategias de arbitraje o plataformas de cambio de forma sencilla siendo además altamente escalable. El sistema, además, permite la generación de gráficos bajo demanda del usuario para la consulta del histórico de beneficios y el estudio del mercado mediante soportes y resistencias. El sistema también está accesible a través de un BOT de Telegram para que el cliente pueda tener acceso a las oportunidades de arbitraje de forma muy sencilla, y que además pueda generar gráficos desde el mismo, cubriendo de esta forma toda la funcionalidad del sistema principal.

Durante el desarrollo de la aplicación han ido surgiendo algunos problemas que se han solventado con éxito, como son la falta de experiencia sobre algunas tecnologías utilizadas (Elixir, Gnuplot) o la implementación de una arquitectura compleja sobre un lenguaje de programación con paradigma funcional. A pesar de todo, el sistema obtenido cumple con el objetivo y las expectivas. Algunos aspectos de mejora a futuro son la implementación de funcionalidades para compra automática, desarrollo de tests automatizados, o la inclusión de más plataformas de cambio o mercados para aumentar las oportunidades de arbitraje.

Índice general

Índice de figuras

Glosario de términos

- **Benchmark**: es un punto de referencia utilizado para medir el rendimiento de una inversión. Se trata de un indicador financiero utilizado como herramienta de comparación para evaluar el rendimiento de una inversión.
- **Billetera**: medio físico o virtual encargado de guardar las llaves públicas/privadas de transacciones de criptomonedas.
- **Blockchain**: registro público y único, consensuado y distribuido en varios nodos de una red en el que sus participantes se conectan entre sí a través de una red punto a punto.
- **Broker**: Persona que, por oficio, actúa como intermediaria en operaciones de compra y venta de valores financieros y de acciones que cotizan en bolsa.
- **Criptomoneda**: medio digital de intercambio que utiliza criptografía para asegurar las transacciones, controlar la creación de unidades adicionales y verificar la transferencia de activos usando tecnologías de registro distribuido.
- **Derivados**: ofrecen a los inversores una forma eficiente de capital de obtener exposición a las criptomonedas sin los tecnicismos para operarlas directamente.
- **Exchange**: plataforma que permite el intercambio entre criptomonedas en base a un precio de mercado que marca el valor de estas criptomonedas según la oferta y la demanda.
- **FIAT**: dinero de curso legal cuyo valor no deriva del hecho de ser un bien físico o mercancía, sino por ser emitido y respaldado por un gobierno.
- **Minería**: es el proceso en el que los mineros utilizan la potencia informática (hash), para procesar transacciones y obtener recompensas, en este caso criptomonedas.
- **Order book**: Lista automatizada de las actuales posiciones de compra y venta para un activo específico.
- **P2P**: red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, formada por una serie de nodos que se comportan como iguales entre sí.
- **Resistencia**: es un precio por encima del actual, la fuerza de venta superará a la de compra, poniendo fin al impulso alcista, y por lo tanto el precio retrocederá.
- **Soporte**: es un nivel de precio por debajo del actual, se espera que la fuerza de compra supere a la de venta.
- **Volumen**: valor de todas las compras y ventas hechas durante un dado período de tiempo, en uno o varios exchanges.

Capítulo 1

Introducción

1.1. Motivación

Este trabajo gira en torno al mundo de las criptomonedas, un tema que se ha popularizado mucho estos últimos años. En estas destacan características muy llamativas siendo la más importante de ellas la descentralización. Esto implica que ninguna entidad bancaria o gobierno tienen control sobre este tipo de divisas. Además, gracias al uso de criptografía (basada en algoritmos hash), las hace muy seguras. Estos factores han provocado que se hayan asentado en nuestra sociedad de tal forma que ya no es extraño oír hablar de ellas. Se prevé que, en los próximos años, este tipo de divisa podría formar parte de nuestro día a día.

La aparición de diversas plataformas para el intercambio de criptomonedas ha provocado la aparición de oportunidades de arbitraje, consistentes en una o varias operaciones financieras que se complementan y permiten obtener beneficio sin que apenas suponga un riesgo. No obstante, dado que es bastante complicado detectar estas situaciones de forma manual, el objetivo de este TFG es el desarrollo de una aplicación que pueda encargarse de tal fin y que, además, permita estudiar de diversas formas el mercado de las criptomonedas, facilitando estas tareas para clientes finales.

Este trabajo supone una oportunidad para aprender más sobre el mundo de las criptomonedas y los mercados financieros, así como el aprendizaje de nuevas tecnologías de las cuales se parte con poca o nula experiencia. Cumplidos todos los objetivos del trabajo, no solo se ha ganado gran experiencia sobre las tecnologías aplicadas y sobre mercados financieros y criptomonedas, sino que, además, se termina con un sistema que da posibilidad de realizar arbitraje en criptomonedas.

1.2. Objetivo

El objetivo de este trabajo es crear una infraestructura software abierta y distribuida para obtener automáticamente oportunidades de arbitraje en criptomonedas, con la finalidad de ganar dinero sin asumir apenas riesgos, que además posea una arquitectura lo suficientemente correcta para posibilitar que otros desarrolladores implementen sobre

ella mecanismos que aumenten las oportunidades de arbitraje, tales son la integración de nuevas plataformas de cambio, de nuevos mercados o incluso nuevas estrategias de cálculo.

1.3. Estructura del documento

Este documento se encuentra dividido en 6 capítulos y un anexo. En el capítulo 2 se introducen conceptos relativos a las criptomonedas, tales como un recorrido por su historia e información sobre su tecnología y funcionamiento, los exchanges y el concepto de arbitraje.. En el capítulo 3 se hace una descripción más detallada del sistema, entrando en detalles más técnicos como es la arquitectura del sistema y todo su funcionamiento, haciendo hincapié en los factores que más se han tenido en cuenta durante la etapa de desarrollo. En el capítulo 4 se mencionan algunos problemas que han surgido durante la realización de este proyecto. El capítulo 5 habla sobre algunas aplicaciones similares que existen actualmente en el mercado. En el capítulo 6 se han sacado algunas conclusiones, mencionando algunos aspectos que se pueden mejorar de cara al futuro.

Al final del documento se encuentra el Anexo A, donde se muestra de forma detallada cómo se ha distribuido el tiempo invertido en este trabajo.

Capítulo 2

Conceptos previos

En este capítulo introducen algunos conceptos que son necesarios para la comprensión del trabajo desarrollado, introduciendo en primer lugar una pequeña recapitulación histórica sobre los acontecimientos más importantes en las criptomonedas, y a continuación se describe cómo es su tecnología y funcionamiento. Después, se mencionan las plataformas de cambio de criptomonedas (*exchanges*), describiendo sus características más importantes. Por último, se habla más en detalle del concepto de arbitraje.

2.1. Historia de las criptomonedas

El origen de las criptomonedas se remonta al 31 de octubre del 2008, donde el criptógrafo Satoshi Nakamoto publicó un documento hablando de la primera criptomoneda moderna y la tecnología que lo envuelve, el Bitcoin [3].

El germen de la idea, por su parte, se sitúa mucho antes. Durante las décadas anteriores han habido figuras que realizaron investigaciones y estudios que servirían de aporte para el desarrollo de esta tecnología tal y como hoy se le conoce. La figura 2.1 representa una línea temporal con los eventos más importantes que desembocan en el surgimiento de las criptomonedas.

David Chaum, conocido como el inventor del dinero digital, publicó un artículo en 1983 hablando sobre sus fundamentos: la utilización de monedas virtuales que fueran anónimas y privadas, que no estén supervisadas ni reguladas por ningún organismo, y que permitiese realizar pagos electrónicos sin que terceros pudiesen determinar información de éste (beneficiario, cantidad, fecha...) [4]. Además, introdujo la primitiva criptográfica de una firma ciega: una forma de firma digital que asegura el contenido de un mensaje antes de que esté firmado, de tal forma, que una vez se firma, el firmante no puede detectar el contenido. A finales de los años ochenta, fundó DigiCash, una compañía de moneda electrónica para producir unidades monetarias basadas en el algoritmo de firma ciega y realizar investigaciones al respecto, que acabó quebrando a finales de los años 90.

La idea de que los puzles computacionales pudieran tener algún tipo de valor fue propuesta por los criptógrafos Cynthia Dwork y Moni Naor en 1992 [5], e investigado por Adam Black en 1998 [6]. Poco después, en 1998, Wei Dai, especializado en software,

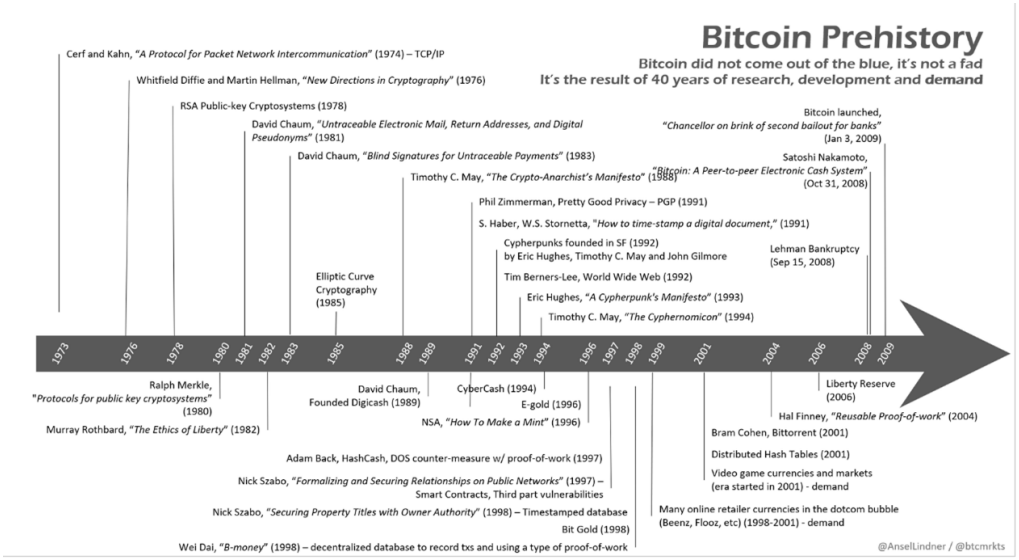

Figura 2.1: Historia de las criptomonedas - Fuente: Reddit.com [1]

comenzó a hablar acerca de una forma de pago que utilizase sistemas criptográficos, proponiendo una solución descentralizada para el problema de los pagos electrónicos [7].

El Bitcoin surge por tanto en base a todo el desarrollo e investigación de años y años, cuidando los detalles y procurando no cometer los mismos errores que se cometieron en el pasado, y surge como la primera criptomoneda moderna. En el año 2010 se inaugura Mt.Gox, el primer *exchange* de Bitcoin. Ubicado en Japón, a los 4 años sufrió un ataque informático que le provocó pérdidas millonarias y le llevaron a su desaparición [8]. A principios del año 2011 se fundó Silk Road, un sitio web para el intercambio de bienes ilegales (en gran parte, drogas), que encontró en el Bitcoin una forma segura y pseudo-anónima para realizar transacciones de este ámbito. A pesar de que empresas con gran vinculación con el Bitcoin se han visto en aprietos (brechas de seguridad, o problemas legales), cada vez es más frecuente el uso de estas criptomonedas por cualquier usuario.

2.2. Criptomonedas: tecnología y funcionamiento

Las criptomonedas son un medio digital de intercambio que utiliza criptografía para asegurar transacciones, controlar la creación de unidades adicionales y verificar la transferencia de activos usando tecnologías de registro distribuido. Son monedas virtuales que pueden ser intercambiadas y usadas como cualquier moneda tradicional, que hace uso de la criptografía, manteniendo así un alto grado de seguridad ya que su falsificación se hace prácticamente imposible.

La descentralización es una de sus principales características, y es algo que las distingue completamente de las monedas tradicionales. Esto quiere decir que no están controladas

por ningún gobierno o entidad financiera, haciéndolas totalmente independientes y totalmente inmunes ante interferencias por parte de terceros. Para llevar a cabo la labor de descentralización, se hace uso de una red P2P, que es un tipo de red en el que cada usuario es al mismo tiempo dueño y contribuidor de la misma, es decir, no existen clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Uno de los usos más conocidos de las redes P2P es el intercambio de ficheros mediante el protocolo BitTorrent.

En el contexto de las criptomonedas, la tecnología *blockchain* usa el poder de las redes P2P para realizar transacciones seguras. *Blockchain* se define como una base de datos en la que cualquiera puede almacenar información, entre las que se incluyen las transacciones de criptomonedas. La estructura de datos en el *blockchain* es muy distinta a la de cualquier base de datos tradicional. Mientras que una base de datos tradicional guarda la información mediante el uso de tablas, el *blockchain* estructura los datos en bloques. Estos bloques poseen cierta capacidad de almacenamiento para añadir cada transacción que se realiza a la red. Cuando el bloque se llena, un nuevo bloque se añade al último bloque llenado.

El procedimiento hace que los datos o la información que se almacena sea imposible de borrar o modificar por ningún tipo de usuario. Esto es gracias a las técnicas criptográficas aplicadas, pues la información contenida en un bloque solo podría ser repudiada o editada modificando todos los bloques posteriores. A su vez, la información está a disposición de todos los usuarios que quieran consultarla.

Al ser descentralizado en una red P2P, en *blockchain* no existe ningún nodo central donde se guarden estos datos. Es decir, cada usuario que forma parte de la red mencionada tiene una copia de todos estos datos, lo que significa que todos los datos del sistema son respaldados por cada nodo que participa en la red.

Existen tres tipos de redes *blockchain*: los de tipo público o *permissionless*, donde cualquier usuario que conforme o no la red puede comprobar las transacciones que se realizan dentro de ésta y participar en el mecanismo de consenso (que se encarga de verificar y validar transacciones y su posterior registro en el bloque); los de tipo privado o *permissioned*, que se caracterizan porque solo puede tener acceso aquellos usuarios que hayan sido previamente autorizados por la entidad gestora de la red o por los integrantes de esa misma red; por último, los *blockchain* híbridos, que surgen de la combinación de los anteriores mencionados.

A pesar de que el Bitcoin ha sido (y aún es) la criptomoneda más relevante y conocida, desde su aparición han ido surgiendo otras como Ethereum, Litecoin o Ripple. En la actualidad, existen más de 9.000 criptomonedas.

Existen diferentes métodos para generar nuevas unidades de criptomonedas, llamados comúnmente procesos de minado. Por ejemplo, monedas como Bitcoin o Litecoin usan *proof-of-work*, que requiere gran poder computacional, mientras que otras monedas como Ethereum usan *proof-of-stake*, que requieren unos criterios preestablecidos de cuotas. La obtención de las unidades existentes se puede realizar a través de los conocidos como *exchanges*.

2.3. Exchanges

Los *exchanges* de criptomonedas son plataformas digitales que permiten el intercambio de monedas digitales por dinero FIAT, otras criptomonedas o incluso mercancías. Suelen ser plataformas centralizadas por lo general y comparten una dinámica automatizada muy similar a otras plataformas de inversión. También es común su uso para almacenar monedas virtuales en sus *wallets*, aunque es poco recomendable, pues se está cediendo la privacidad y la seguridad a un tercero. Además, los *exchanges* suelen ser objetivo de cibercriminales. Para aquellas personas que desean comprar criptomonedas y no pueden optar a realizar procedimientos de minado, es la opción perfecta y como es obvio, la más usada para la obtención de criptomonedas.

El funcionamiento de estas plataformas se basa en emparejar compradores y vendedores, como sucede en cualquier mercado de valores. Los compradores establecen la cantidad que quieren adquirir de cierta moneda mediante una orden de compra. Si hay una orden de venta del mismo precio, entonces el *exchange* realiza la pareja entre estas dos órdenes y se realiza el cambio. Existen varios tipos de *exchanges*:

- **Brokers:** a pesar de no ser *exchanges* de criptomonedas puramente, pueden actuar entre intermediarios entre los mercados de criptomonedas y los inversores que deseen comprar o vender este tipo de material digital. Cualquiera puede visitar un bróker y comprar criptomonedas al precio que esté determinado.

- **Exchanges tradicionales:** facilitan la compra y la venta de monedas digitales basado en los precios diarios del mercado. Normalmente, junto con las transacciones, establecen algunas comisiones.

- **Exchanges cetralizados:** se trata de un tipo de *exchange* controlado por un tercero (al que se le denomina operador) y que ayuda a que el alta y las transacciones funcionen correctamente. Destacan porque hacen sencillo y rápido enlazar información bancaria o tarjetas de crédito para la compra de criptomonedas, aunque hay que tener en cuenta que esto lleva consigo algunas comisiones.

- **Exchanges descetralizados:** puesto que las criptomonedas y el *blockchain* fueron diseñadas con la filosofía de que se debería funcionar sin ninguna autoridad central que controle los movimientos, aparecen este tipo de plataformas, de código abierto, que hacen uso de la ya conocida red P2P. Al no existir un servidor central, es complicado comprometer la seguridad, por lo que se convierten en mucho más seguras que las plataformas centralizadas. No obstante, el uso de estos tipos de *exchanges* requieren usualmente altos conocimientos informáticos y en el propio funcionamiento de las criptomonedas.

Derivada de la popularidad de las criptomonedas en los últimos años, han surgido muchos *exchanges* que ofrecen características ventajosas para atraer a inversores. Sin embargo, hay distinguir varios factores para elegir un buen *exchange* como, por ejemplo: los límites de depósito y retirada de dinero, las comisiones (tanto de depósitos y retiradas,

como a la hora de realizar operaciones), formas de pago, las regulaciones y restricciones que se aplican, la reputación que mantienen a nivel general, y la verificación de identidad que realizan.

Los *exchanges* también se han convertido en un imán para cibercriminales, derivado del valor que han obtenido las criptomonedas en los últimos años y el pseudo-anonimato que las respaldan, siendo protagonistas de robos millonarios a lo largo de esta última década. Ejemplos destacados son el ataque registrado en el *exchange* Coincheck, donde se robaron 523 millones de la criptomoneda NEM, equivalente a 530 millones de dólares y afectando a 260.000 usuarios [9] y el robo al *exchange* japonés Mt. Gox , que supuso la pérdida de 473 millones de dólares [8]. Estos ataques continúan en la actualidad, tan solo en el año 2020 se produjeron 28 ataques satisfactorios, suponiendo la pérdida de más de 300 millones de dólares [10].

2.4. Arbitraje

El arbitraje es un término que hace referencia a la práctica de sacar provecho entre una diferencia de precio entre dos o más mercados [11]. Se realiza una combinación de transacciones (complementarias entre sí) logrando cubrir el desequilibrio de precios, pudiendo obtener un beneficio sin la necesidad de arriesgar capital. Este tipo de operaciones no dependen de las tendencias de mercado alcistas o descendientes y es posible ganar dinero independientemente del rumbo que tome un determinado mercado. Las personas que se dedican a realizar este tipo de tareas se conocen como *arbitrajistas*.

Para que el arbitraje sea posible, debe cumplirse algunas de las siguientes condiciones: `(i)` que el activo no se transmita al mismo precio en distintos mercados; `(ii)` que no se transmitan al mismo precio dos activos que producen el mismo flujo de efectivo; o `(iii)` que un activo con un precio conocido en el futuro no se vende hoy a su precio futuro descontado a la tasa de interés libre de riesgo.

No solo se trata de comprar un producto en un mercado y venderlo en otro mercado a un precio más alto, sino que dichas transacciones de compra y venta deben ocurrir en el mismo instante de tiempo (o un instante de tiempo muy pequeño) para evitar exponerse a los riesgos de mercado. A pesar de que con la existencia de mercados electrónicos los riesgos son bajos (pues facilitan que dos transacciones puedan realizarse casi al mismo tiempo), no garantizan que las operaciones se cierren exactamente en el mismo instante. Esto puede provocar que cuando una de las operaciones ya se ha cerrado, en ese momento se produzca un rápido cambio en los mercados de la(s) otra(s) operaciones, provocando que se pierda la oportunidad de arbitraje. Este factor hay que tenerlo especialmente en cuenta, pues para obtener cierta rentabilidad se tiene que operar con gran cantidad de dinero, por lo que si el trato no se consigue cerrar por alguna de las dos partes puede suponer la pérdida de una gran cantidad de dinero.

Otros factores de riesgo a tener en cuenta son los bienes que se compran, ya que se debe asegurar que sean idénticos y a la vez, realizar operaciones que sean complementarias. Si se realizan operaciones que no son complementarias, puede desembocar también en pérdidas desastrosas.

Existen más factores por los cuales ocurren estas oportunidades de arbitraje, como es la varianza de liquidez entre varios *exchanges*, pues la gran mayoría de ellos tienen su propio *order book*: si un *exchange* tiene un *order book* más amplio y la otra uno más lleno, sería prudente que se comparara el activo en la última, ya que en la primera se terminaría pagando un precio más alto. Los tiempos de depósitos y retiradas también son un factor bastante importante. Aquellos *exchanges* con mayor velocidad a la hora de realizar transferencias se emparejan con el mercado antes, mientras que otros se van adaptando más lentamente.

Algunos aspectos que además se deben tener en cuenta son los siguientes:

- Las comisiones de transacciones: la gran mayoría de *exchanges* suelen aplicar comisiones a las operaciones que realizan sus clientes. Estas comisiones hay que tenerlas en cuenta a la hora de realizar cualquier operación de arbitraje, ya que supone una minimización en los beneficios que otorga esa operación.
- Acciones de los *exchanges* impredecibles: estas plataformas tienen el control sobre las operaciones. Esto supone que un *exchange* podría tomar la decisión de limitar las transacciones para una criptomoneda, o incluso cerrar el mercado de forma temporal por varias circunstancias, como pueden ser periodos de muy alta volatilidad.
- La tecnología: se necesita algún tipo de software que de forma automatizada pueda detectar oportunidades de arbitraje. Los mercados fluctúan muy rápido y puede ser muy complicado para una persona obtener oportunidades de forma manual. La rapidez en este mundo es tremendamente importante.
- Regulaciones: algunas regulaciones como KYC (*Know Your Client*) o AML (*Anti-Money Laundering*) pueden impactar en las operaciones de arbitraje.

Para realizar arbitraje, existen varias estrategias. La más básica de ellas, conocida como estrategia espacial, consistiría simplemente en comprar en un *exchange* a un precio bajo, para transferir a otro *exchange* y vender a un precio más alto. Pero existen otras estrategias más sofisticadas y complejas, que a su vez pueden otorgar mayores beneficios, como es la estrategia triangular donde se realizan 3 operaciones. Se hablará de ellas con mayor detalle en la sección 3.3.3.

Capítulo 3

Sistema desarrollado

En este capítulo se describe en detalle el sistema desarrollado, comenzando con una visión arquitectural, la cual trata de facilitar su comprensión con ayuda de varios tipos de diagramas. También se comentan los aspectos importantes en cuanto a la tecnología utilizada y se realiza una descripción a nivel algorítmico de las múltiples funcionalidades del sistema.

3.1. Requisitos del sistema

Las necesidades expuestas en los apartados anteriores convergen en la creación de una infraestructura software que sea capaz de detectar oportunidades de arbitraje de forma automatizada, aplicando diversas estrategias, que esté construido sobre una arquitectura distribuida que permita escalabilidad y que además haga sencillo la agregación de nuevas estrategias o plataformas sobre las que realizar arbitraje.

Para tener claro lo que se procede a diseñar, se realiza una lista muy general de requisitos funcionales y no funcionales.

- Requisitos funcionales (RF):

RF1. El sistema permitirá obtener oportunidades de arbitraje en base a la información proporcionada por uno o varios *exchanges*.

RF2. El sistema permitirá calcular oportunidades de arbitraje mediante varias estrategias al mismo tiempo.

RF3. El sistema permitirá visualizar el histórico de beneficios entre *exchanges* para un mercado concreto mediante un gráfico, en un rango de tiempo determinado por el usuario.

RF4. El sistema permitirá realizar el cálculo de soportes y resistencias para un mercado en concreto en un rango determinado de tiempo, y visualizarlo mediante un gráfico.

RF5. El sistema permitirá mostrar la información calculada y generada por la aplicación mediante una API Rest en un servidor Web.

RF6. El sistema posibilitará al cliente darse de alta en un BOT de Telegram y recibir información periódica acerca de las oportunidades de arbitraje.

RF7. El sistema posibilitará al cliente configurar sus preferencias en el BOT de Telegram (mercados, *exchange*s a monitorizar, etc.).

- Requisitos no funcionales (RNF):

RNF1. El sistema permitirá ser desplegado de forma distribuida.

RNF2. El sistema permitirá despliegue arbitrario de forma sencilla.

RNF3. El sistema debe ser tolerante a fallos.

RNF4. El sistema permitirá realizar la operativa a la mayor brevedad posible, minimizando las latencias y los tiempos de procesado tanto como sea posible.

La aplicación concentra su funcionalidad principal en hacer un sistema distribuido y tolerante a fallos que permita el cálculo de oportunidades de arbitraje. No obstante, debido al potencial que tiene el hecho de trabajar con varios mercados y varios *exchange*s, se puede aprovechar toda la información que se dispone y añadir más funcionalidades al sistema, como es la generación de gráficos para el histórico de beneficios y también los cálculos de soportes y resistencias para estudio de mercado.

3.2. Arquitectura

Con el fin de entender mejor la arquitectura de la aplicación se han realizado diagramas UML (*Unified Modeling Language*) [12]. UML es de un estándar que permite describir un sistema en forma de "plano". Se incluyen en él aquellos aspectos tales como funcionalidad del sistema, expresiones del lenguaje concreto, etc. Este estándar utiliza y asocia elementos de distintas formas con el fin de desarrollar diagramas que representen ciertos aspectos tanto estructurales como estáticos de un sistema en concreto, como la colaboración de los distintos objetos de este de forma específica.

Se ha diseñado un diagrama de clases (para tener una visión global de todo el sistema junto con todos sus componentes), diagramas de secuencia (para comprender la interacción entre todos los componentes implicados en el sistema), y un diagrama de despliegue (para apreciar cómo queda la arquitectura una vez está desplegada).

Para el propio diseño de la arquitectura se ha hecho especial hincapié en que debe ser lo suficientemente desacoplada como para añadir nuevas plataformas de cambio y nuevas estrategias de cálculo de arbitraje sin la necesidad de modificar el núcleo de la infraestructura.

Tal y como muestra la figura 3.1, la arquitectura está conformada por varios paquetes. El paquete principal de la arquitectura es el paquete *Core*. Este paquete mantiene toda la lógica interna de la infraestructura y conforma aquellas partes cuya modificación no es necesaria para añadir componentes o funcionalidad al sistema. En este paquete se encuentra toda una serie de clases, que heredan todas de la clase *DistributedModule*.

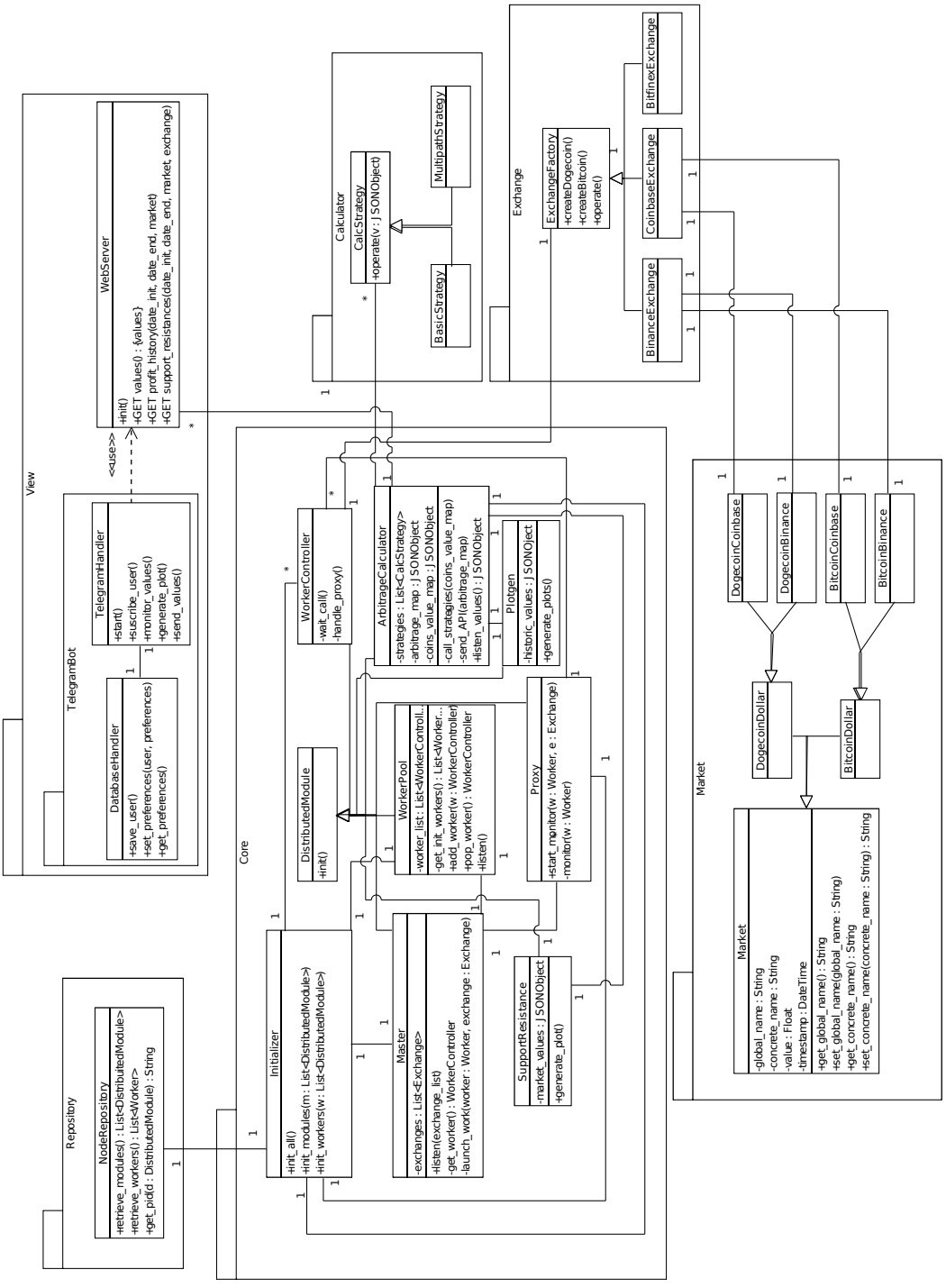

Figura 3.1: Diagrama de clases

Esto es así porque cada una de las clases de este módulo van a representar un proceso en ejecución distribuido.

Existe otro paquete *Repository*, que mantiene una clase *NodeRepository* utilizada como interfaz para establecer conexiones a la base de datos, y tiene que ver con la operativa para distribuir el sistema.

También está el paquete de vistas para todo aquello relativo a dar visibilidad al sistema. Éste tiene un servidor web a través del cual se expone toda la información que se calcula.

En el paquete de *Calculator* quedan reflejadas todas las estrategias de cálculo para arbitraje disponibles. Se utiliza un patrón *Strategy* [13]. Este patrón permite establecer varios comportamientos ante un evento (cálculo de arbitraje) de una forma sencilla, de tal forma que integrar nuevas estrategias de cálculo (dado un conjunto de datos sobre los cuales hacer los propios cálculos) resulta una tarea sencilla.

Para tareas de obtención de valores en plataformas de intercambio de criptomonedas, hay que tener en cuenta que cada una puede tener su propia estrategia y que, además, el nombre de referencia para los mercados en cada uno es distinto. Por ello, este punto era crítico en cuanto a la arquitectura. Para solucionarlo se ha decidido aplicar un patrón *Abstract Factory* [13] donde existe una factoría de *exchange* para generar los propios *exchanges* y en cada uno se generan a su vez los mercados que se van a monitorizar, teniendo en cuenta que se va a disponer de una serie de mercados registrados en el sistema (globales) y que en cada *exchange* exiten nombres distintos. Para capturar estas peculiaridades de los mercados se necesita un objeto *Market*, y de esta forma, crear mercados necesarios (abstrayéndolo tal como se muestra en el diagrama) para cada uno de ellos.

3.3. Funcionamiento y características

Para el desarrollo del sistema se ha tomado la decisión de utilizar la tecnología Elixir. Se trata de un lenguaje de programación relativamente reciente con un paradigma funcional y concurrente, que además está desarrollado y se ejecuta sobre la máquina virtual de Erlang y fue diseñado para crear aplicaciones escalables y mantenibles.

Sus características de escalabilidad, concurrencia y tolerancia a fallos, entre otras, hacen que este lenguaje se adapte muy bien a todo lo que requiere esta aplicación. Téngase en cuenta que se requiere que el sistema mantenga bajas latencias para las oportunidades de arbitraje y rapidez en todo el procedimiento, desde que se obtienen los datos hasta que llegan al cliente.

El sistema debe estar pensado para que pueda trabajar con gran cantidad de datos. Hay que suponer que se tendrán muchos *exchanges* y muchos mercados obteniendo y procesando valores cada unos pocos milisegundos. Elixir y otras tecnologías basadas en Erlang son capaces de rendir eficazmente con grandes volúmenes de datos.

En la figura 3.2 se realiza una comparativa, en donde 3 lenguajes recientes son sometidos a 100.000 conexiones simultáneas mostrándose su comportamiento. Elixir, junto con Go, pueden atender todas las peticiones sin mayor problema, pero se aprecia sobrecarga

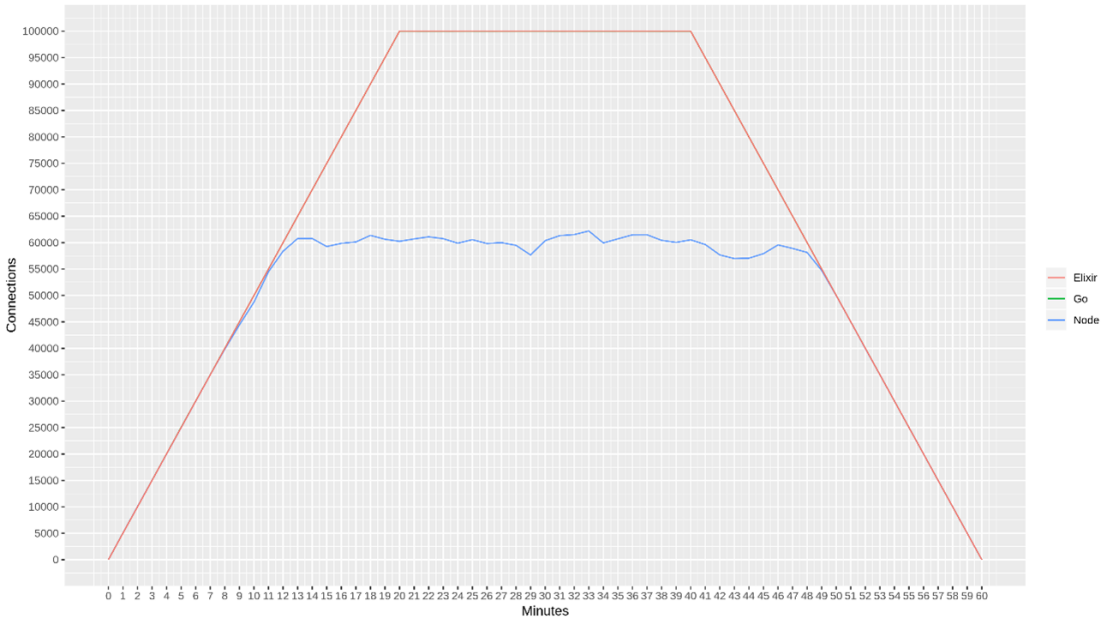

Figura 3.2: Benchmark Go vs. Node vs. Elixir - Fuente: Stressgrid.com [2]

en Node a partir de las 60.000 conexiones.

También se debe tener en cuenta el factor de la escalabilidad. La forma en que está diseñado Elixir permite convertir un proceso en distribuido de una forma muy sencilla e identificarlo de forma sencilla para permitir la comunicación entre varios nodos, lo que favorece en la creación de un sistema distribuido, y también en la escalabilidad.

La tolerancia a fallos en un sistema que pretende estar operando a pleno rendimiento las 24 horas del día y donde la información que se obtiene ha de ser coherente (no puede haber inconsistencia), es un factor importante. A pesar de que Elixir trae consigo mecanismos para tolerancia a fallos, estos son algo escasos por lo que habrá que implementar algunos mecanismos adicionales para garantizar esta funcionalidad del sistema.

Respecto al BOT en Telegram, se ha tomado la decisión de utilizar Python. De esta forma, se puede demostrar el desacoplamiento de la aplicación, mostrando así que no se depende del código de la aplicación general para su funcionamiento y que a través de una Rest API se permite obtener la información. Además, Python tiene la característica de que funciona muy bien para tareas de este tipo.

3.3.1. Funcionamiento

En este apartado se explica de forma detallada el funcionamiento del sistema desarrollado. Con el fin de comprenderlo mejor, se ha diseñado un diagrama de secuencia que visualiza la funcionalidad principal del sistema, que tiene que ver con la obtención de oportunidades de arbitraje y tolerancia a fallos.

Todo el sistema está conformado por una serie de procesos. Se distingue, por una parte, los Módulos, que son aquellos procesos que representan las partes fundamentales del sistema. Por otra parte, los workers son aquellos procesos encargados exclusivamente de ir recopilando valores de los *exchanges*. En este sistema, por tanto, existirá un worker por cada *exchange*.

La figura 3.3 muestra cómo es la interacción global del sistema. Toda la interacción la comienza un proceso Inicializador, llamado *Initializer*. Nada más lanzar el *Initializer*, se realizan algunas comprobaciones (como la arquitectura o el sistema operativo). Justo después se realiza la primera interacción en el repositorio de Nodos. El sistema, al ser distribuido, debe permitir que varias partes del sistema estén desplegadas en varios nodos distintos. Se necesita determinar la forma en que los nodos conozcan qué módulos o workers debe ejecutar y cuáles no. Se ha tomado la decisión de utilizar una base de datos centralizada MySQL donde se incluyesen todos los módulos y workers del sistema, así como los nodos que se disponen, y relacionarlos entre ellos para determinar qué se debe ejecutar en cada uno.

En la figura 3.4 se muestra un diagrama entidad-relación de la base de datos en cuestión, conformada de tres tablas. En una de ellas, llamada *hosts* se guardan todos los nodos en los que se van a desplegar las distintas partes del sistema. Concretamente se guarda el nombre del sistema o *hostname*, que es la forma con la que se identifican los nodos en el momento del despliegue. Las otras dos tablas se usan para guardar los workers y módulos, guardando de cada uno el nombre concreto y la dirección con la que se debe referir a ellos. En Elixir, todos los procesos que se convierten en distribuidos se identifican por un nombre y una dirección. La estructura de este último tiene un formato del tipo `nombrenodo@ip`. Por ejemplo, para el módulo *Master* quedaría como `master@127.0.0.1`, suponiendo que se está desplegando en máquina local.

Por tanto, nada más ejecutarse, a través de la interfaz *NodeRepository* se hace una consulta a la base de datos centralizada `get_modules`, con su *hostname*, para obtener los módulos y los workers `get_workers` que deben ser desplegados en su máquina. Una vez que se conoce qué procesos se deben ejecutar, los lanzará `initialize`, se convertirán en distribuidos, habilitando la posibilidad de que puedan dar y recibir órdenes remotas.

Toda la aplicación se sustenta bajo una arquitectura Máster-Worker. El módulo principal, por tanto, será el Máster que es el encargado de coordinar el resto. La primera tarea del Máster será obtener la lista de *exchanges* que debe monitorizar. Para cada uno de los *exchanges*, deberá haber un worker disponible para hacer la tarea de obtención de valores. Una vez que tiene los *exchanges* que debe monitorizar, el siguiente paso es comunicarse con el pool de workers mediante `request_worker`.

El módulo *WorkerPool* es el encargado de mantener todos los workers que hay en el sistema, saber cuál es su estado (disponibles u ocupados), y proporcionar al Máster las direcciones para que pueda comunicarse con ellos según éste se las pida. Nada más iniciar este módulo, lo que hace es consultar a la base de datos a través de la interfaz *NodeRepository* qué workers hay en el sistema mediante `get_workers`. Cuando el Máster requiera al pool de workers un nuevo worker, va a obtener el primer worker disponible, va a comprobar que realmente el worker está disponible con `check_alive` y responde

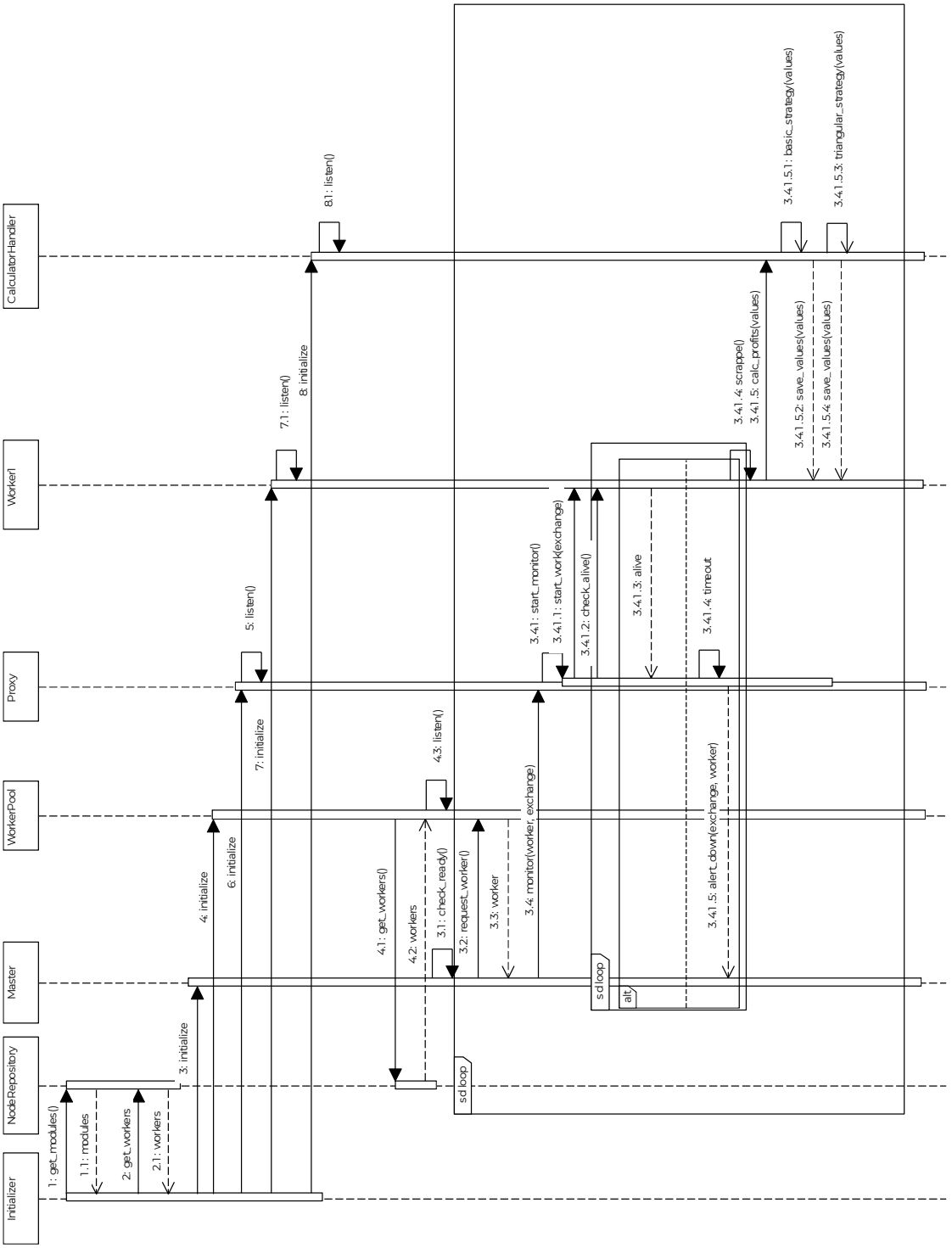

Figura 3.3: Diagrama de secuencia general

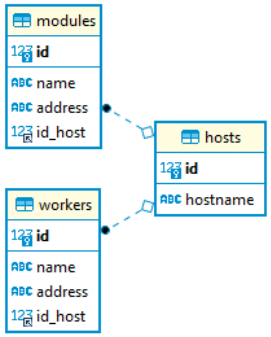

Figura 3.4: Diagrame Entidad-Relación para despliegue

(para evitar inconsistencias). Si está disponible, automáticamente comunicará al Máster la dirección para conectarse. Si al comprobar su disponibilidad han pasado más de 5 segundos y todavía no ha obtenido respuesta, ese worker se considera caído y pasará a realizar la misma operación con el siguiente worker. Como mecanismo de recuperación ante workers caídos, se abre un hilo donde se le enviará periódicamente señales para comprobar si responde de alguna forma. Cuando vuelva a responder, se informará al pool de workers de que el worker está vivo para que lo vuelva añadir a su lista de workers.

En caso de que el Máster requiera un worker y no haya más disponibles, el pool de workers se lo hará saber y esperará un tiempo arbitrario para volver a intentarlo, hasta que el pool de workers le proporcione finalmente un worker para trabajar. Se asume por tanto que para un buen funcionamiento del sistema deberían de haber tantos workers como *exchanges* se deban analizar.

En este momento, el Máster ya ha conseguido un worker y conoce los *exchanges* con los que tiene que trabajar. Con el fin de mejorar la tolerancia a fallos, no se realiza una interacción directa entre Máster y workers si no que se realiza mediante un *Proxy*. La tarea de este *Proxy* será dar la orden al worker para que comience a trabajar y comprobar periódicamente que el worker sigue vivo. Por cada uno de los workers, el *Proxy* va a crear un hilo mediante el método `start_monitor`. Este hilo da orden al worker de que comience la operativa de escaneo de valores para un *exchange*. El worker por su parte, tiene un hilo principal encargado de la tarea de escaneo de valores (`scrappe`), pero tendrá un hilo secundario encargado de enviar latidos de corazón al *Proxy* (método `check_alive`) cada cierto tiempo para que éste compruebe que sigue vivo.

En cuanto a la funcionalidad principal de los workers, el procedimiento que siguen es el siguiente:

1. Llamar a la API de un *exchange* usando el método `scrappe`
2. Recoger y paresear los datos
3. Enviar los datos a la calculadora de arbitraje `calc_profits`

Este procedimiento se repite por cada uno de los mercados. Como ya se mencionó en el apartado de arquitectura, existe un problema que tiene que ver con el nombre de los mercados difiere entre *exchanges*. Por ejemplo, en Binance no existe el mercado BTC/USD, sino que está el mercado BTC/USDT, que a su vez son equivalentes. USDT (Tether) se trata de una *stablecoin*, diseñada para mantener el mismo valor con respecto al dólar estadounidense. La existencia de estas monedas virtuales a veces resulta útil, ya que los inversores pueden evitar la volatilidad de mercado que existen en otras criptomonedas. Además, elimina costes extra y retrasos para las conversiones entre criptomonedas y monedas FIAT.

Por tanto, se establecen relaciones entre un nombre genérico y el nombre concreto de un *exchange*, mediante el uso de factorías, para que a la hora de realizar los cálculos de arbitraje no haya problemas.

3.3.2. Exchanges seleccionados

Para este trabajo se han seleccionado 3 *exchanges*:

- Binance [14]: fundada en 2017 por Changpeng Zhao, dispone de más de 100 mercados de criptomonedas con los que operar, siendo además la plataforma con mayor volumen comercial del mundo. Posee incluso su propia criptomoneda: BNB.
- Bitfinex [15]: fundada en 2012 por la empresa iFinex.
- Kraken [16]: se propuso como reemplazo de la plataforma Mt. Gox tras su enorme brecha de seguridad.

Entre las razones por las que se han seleccionado estos *exchanges* destacan sus bajas comisiones (factor fundamental para este tipo de operaciones ya que requieren mover mucho dinero), que disponen de una amplia lista de mercados (por lo que aumentan las posibilidades de arbitraje) y que proporcionan además una API sencilla sobre la que trabajar.

Es importante también conocer que las APIs tienen ciertas restricciones. Lo ideal sería poder enviar y recibir peticiones cada unos pocos milisegundos y con unas latencias lo más bajas posibles (<25ms). No obstante, esto no es posible. Si se intenta lanzar peticiones cada pocos milisegundos, los servidores terminan por denegar las peticiones puesto que lo consideran un posible ataque de denegación de servicio. Por ello, tras hacer varias pruebas, se determinó que las APIs permitían una tolerancia de unos 500ms entre petición y petición. No obstante, también se implementaron mecanismos de recuperación en caso de que las peticiones comiencen a fallar, independientemente de la causa, pues aparte del hecho de que el servidor establezca restricciones por realizar demasiadas peticiones puede haber caídas temporales de los servicios. Por tanto, en estos casos se hace una espera de tiempo arbitraria y se vuelve a intentar la petición.

También es importante que cuando se obtiene los valores de mercado de cada *exchange* haya un mínimo de sincronismo. Puesto que después habrá que comparar estos valores, es necesario que estos valores estén tomados en el mismo instante. De nada sirve comparar

dos valores de mercado para realizar arbitrajes cuando la toma de estos valores difiere en más de 15 segundos. Por ello se decidió añadir a cada valor que se tomaba una marca de tiempo y así hacer un pequeño estudio sobre los retrasos existentes. Al principio, al comenzar el sistema a operar en cada *exchange* al mismo tiempo, los retrasos son mínimos (apenas unos milisegundos). Sin embargo, debido a los retrasos de cada una de las APIs y otras causas ajenas al sistema desarrollado, los retrasos pueden dispararse hasta los 3-4 segundos entre las tomas. No obstante, al ser un proceso cíclico y tener un tiempo de refresco bajo, será muy complicado que los retrasos alcancen estos máximos durante un tiempo prolongado de funcionamiento.

Por último, lo que se hace es enviar los datos capturados al módulo *ArbitrageCalculator* donde se van a realizar todas las operaciones pertinentes para obtener las oportunidades de arbitraje. Este módulo va a mantener una lista con todos los valores de cada *exchange* actualizados y a la vez, una estructura con todos los beneficios de varias estrategias entre varios mercados.

Cada vez que un worker envía un valor a este módulo, va a añadir el valor de mercado a su lista e inmediatamente va a lanzar N hilos (siendo N el número de estrategias que hay en el sistema) para calcular oportunidades de arbitraje.

3.3.3. Estrategias

Existen varias formas de proceder a la hora de hacer arbitraje. Para este sistema se ha tomado la decisión de cubrir dos de las principales: la estrategia básica o espacial, y la estrategia triangular.

3.3.4. Estrategia básica

Es la estrategia más común y a la vez la más sencilla. El mecanismo consiste en realizar una compra de determinada criptomoneda en un *exchange* a un precio bajo y transferir a otro *exchange* para vender a un precio más alto. En la teoría puede parecer una buena estrategia, pero algunos problemas como los tiempos de transferencia pueden hacer que esta estrategia no sea tan beneficiosa.

A continuación, se describe su funcionamiento. En el sistema, teniendo ya una estructura con todos los valores de los mercados de todas las *exchanges* administrada por el módulo *ArbitrageCalculator*, se realizan comparaciones entre todos los pares de *exchanges*. Suponiendo que se disponen de tres plataformas en el sistema (Binance, Bitfinex, Kraken), las comparaciones quedarían de la forma: Binance-Bitfinex, Binance-Kraken y Bitfinex-Kraken.

Se recorren ahora todos los mercados de cada par de *exchanges*, comparando cada uno de los valores de mercado, determinando las oportunidades de arbitraje. El cálculo del beneficio una vez que se tienen los valores resulta bastante sencillo. Tan solo se debe comparar los valores de cada mercado y determinar cuál es el mayor y cuál es el menor. Esto determinará en qué *exchange* ha de realizarse la compra (valor menor; v_L) y en qué *exchange* ha de realizarse la venta (valor mayor; v_H). Sabiendo esto, solo hay que aplicar la fórmula para obtener el beneficio p como $p = v_H/v_L$.

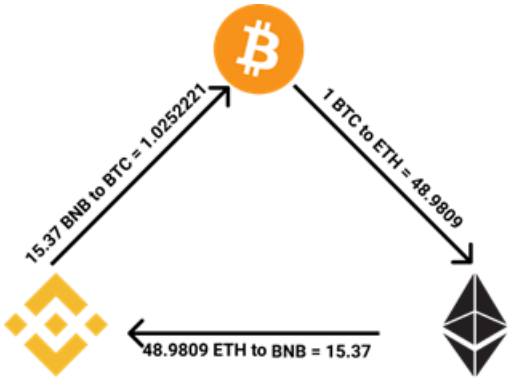

Figura 3.5: Ejemplo de aplicación estrategia triangular

Para entender mejor la forma de proceder, se explica a continuación con un ejemplo. Supóngase que en el *exchange* A el valor de mercado BTC/USD es de 33.656 dólares, mientras que en el *exchange* B el valor de mercado BTC/USD es de 35.712 dólares. El *exchange* A sería donde se debería hacer la compra y el *exchange* B donde se debería vender. Aplicando la fórmula para obtener el beneficio, resultaría $35{,}712/33{,}656 = 1{,}061$.

Se obtiene un beneficio de 1,061. Esto significaría que, al completar las operaciones de arbitraje, se obtendría 0,061 unidades por cada unidad monetaria invertida. En caso de que el resultado fuera menor o igual que 1, significaría que no hay oportunidad de arbitraje y que se podría perder dinero en las operaciones.

3.3.5. Estrategia triangular

Esta estrategia es una estrategia más compleja que el anterior. Es posible realizarla cuando el precio entre tres criptomonedas distintas difiere, en un *exchange* o entre varios *exchanges*, teniendo en cuenta que el triángulo debe terminar con la misma moneda que con la que se comenzó.

La figura 3.5 muestra un ejemplo de cómo se procede para realizar esta estrategia. Primero se realiza la compra de BTC/ETH en un determinado *exchange*, por el cual se obtiene la suma de 48.98 ETH. A su vez, se hace la compra en otro mercado (ETH/BNB). Por la cantidad de Ethereum que se disponen, se obtienen 15.37, los cuales finalmente se pasarán a Bitcoin (BNB/BTC), siendo esta la moneda con la que se comenzó. Al finalizar todas las operaciones, se habrán obtenido 1.025 Bitcoin frente a los 1 Bitcoin con los que se comenzó, lo que supone un beneficio de 0.025 Bitcoin.

A diferencia de la estrategia espacial, donde las oportunidades de arbitraje son más comunes, encontrar una oportunidad con esta estrategia resulta algo poco casual. Esta estrategia resulta bastante más complicada de implementar que la estrategia básica. Hay que tener en cuenta que para los mercados de las criptomonedas se dispone de una peculiaridad. La gran mayoría de las *altcoin* solo admiten dólar o Bitcoin como moneda de cambio (algún *exchange* también puede admitir Ethereum o Litecoin, pero es poco

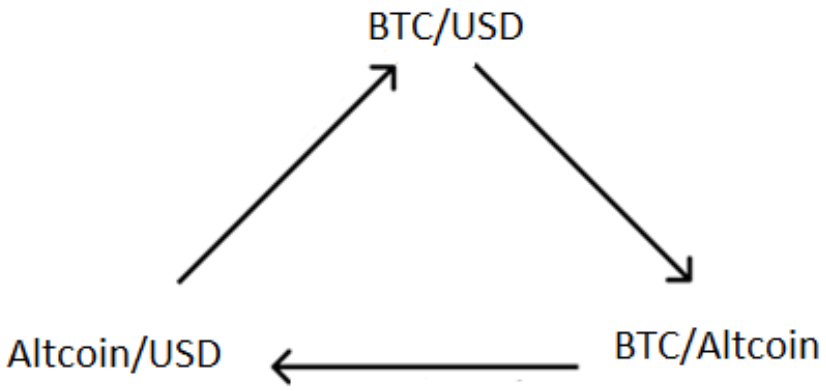

Figura 3.6: Aplicación de estrategia triangular en el sistema

común). No existen los mercados en base a otras *altcoins*, es decir, si se dispone de la moneda TRX y se desea comprar ADA, no se podrá hacerlo directamente, pues no existe ningún mercado TRX/ADA, pero sí existe un mercado ADA/BTC o ADA/USD, lo que implica que habrá que convertir la moneda antes.

Esto tiene unas implicaciones para el algoritmo triangular, pues implicará que en uno de los lados del triángulo tendrá que existir el mercado BTC/USD y en los otros dos lados se tiene una compra/venta con una Altcoin/Dólar o Altcoin/Bitcoin, quedando algo tal y como se muestra en la 3.6.

Esta anomalía limita la probabilidad de oportunidades de arbitraje pero al mismo tiempo simplifica el algoritmo. Para este algoritmo, al mismo tiempo, se ha de que comparar los valores de mercado de cada *exchange* entre sí: mientras que en el anterior algoritmo se formaban pares de *exchanges* distintos y sobre ello se comparaban los valores, aquí sí que tiene sentido, ya que al recorrer más de dos mercados diferentes, es posible tener una oportunidad de arbitraje sin siquiera tener que transferir entre *exchanges*. Por tanto, surgirían las combinaciones (agrupadas de tres en tres) de forma exponencial: Binance-Binance-Binance, Binance-Binance-Bitfinex...

Conociendo esto, lo único que queda es ir recorriendo cada combinación de *exchanges* y cada combinación de mercados de forma iterativa. El algoritmo consiste en un bucle anidado de 3 niveles (técnicamente son funciones recursivas, por limitaciones del lenguaje Elixir), donde el primer nivel del bucle representaría el mercado BTC/USD, el segundo nivel representa el mercado USD/Altcoin y el tercer nivel, el mercado BTC/Altcoin. De esta forma se van a recorrer todas las combinaciones de mercados de cada uno de los *exchanges*.

En cada iteración, una vez que se disponen de las órdenes del mercado correspondiente de cada *exchange*, basta con aplicar la siguiente fórmula.

$$Beneficio = USD - X/(X - BTC * BTC - USD)$$

Supóngase que se desea comprobar el beneficio con la *altcoin* DOGE, y que los valores

de los mercados DOGE/USD es de 0.32, el de DOGE/BTC es de 0,0000089 y el de BTC/USD es de 35462 dólares. Ahora se aplica la fórmula: $0{,}311/(0{,}00000089 * 35462) = 1{,}014$

Realizar las operaciones correspondientes para este caso, supondría obtener un beneficio de 0,014 unidades por cada unidad monetaria invertida, siendo el significado de este valor el mismo que el del apartado anterior.

Cabe destacar que una vez que se obtienen los valores de cada mercado, el orden de ejecución de las operaciones es indiferente. Es decir, si dados 3 valores de 3 mercados, al ejecutar las órdenes de compra/venta en el orden de vértices 1 – 2 – 3 se obtiene el mismo beneficio que si se ejecutan en el orden 2 – 3 – 1 o que en el orden 3 – 1 – 2.

Al igual que en la estrategia anterior, cuando se descubre un beneficio positivo, es enviado al módulo *ArbitrageCalculator* para que actualice su estructura de datos clave valor. La información se transmite en formato JSON, teniendo el siguiente formato:


```
{
   "basic":{
      "ADA_BTC":{
         "binance-bitfinex":{
            "max_exchange":"bitfinex",
            "max_value":3.946e-5,
            "min_exchange":"binance",
            "min_value":3.945e-5,
            "profit":1.0002534854245881
         },
         "binance-kraken":{
            "max_exchange":"kraken",
            "max_value":3.948e-5,
            "min_exchange":"binance",
            "min_value":3.945e-5,
            "profit":1.0007604562737644
         },
         "bitfinex-kraken":{
            "max_exchange":"kraken",
            "max_value":3.948e-5,
            "min_exchange":"bitfinex",
            "min_value":3.946e-5,
            "profit":1.0005068423720225
         }
      },
      "ADA_USD":{
         "binance-bitfinex":{
            "max_exchange":"bitfinex",
            "max_value":1.406400119859,
```

```
            "min_exchange":"binance",
            "min_value":1.40318698,
            "profit":1.002289887167425
         },
         "binance-kraken":{
            "max_exchange":"kraken",
            "max_value":1.405241,
            "min_exchange":"binance",
            "min_value":1.40318698,
            "profit":1.0014638248710088
         },
         "bitfinex-kraken":{
            "max_exchange":"bitfinex",
            "max_value":1.406400119859,
            "min_exchange":"kraken",
            "min_value":1.405241,
            "profit":1.0008248548533667
         }
      }
   },
   "triangular":{
      "ADA_BTC":{
         "binance-binance-binance":{
            "btc_usd_exchange":"binance",
            "btc_usd_value":35581.83134952,
            "profit":0.9996322109108438,
            "usd_x_exchange":"binance",
            "usd_x_value":1.40318698,
            "x_btc_exchange":"binance",
            "x_btc_value":3.945e-5
         },
         "binance-binance-bitfinex":{
            "btc_usd_exchange":"binance",
            "btc_usd_value":35581.83134952,
            "profit":0.9993788829303797,
            "usd_x_exchange":"binance",
            "usd_x_value":1.40318698,
            "x_btc_exchange":"bitfinex",
            "x_btc_value":3.946e-5
         },
         "binance-binance-kraken":{
            "btc_usd_exchange":"binance",
            "btc_usd_value":35581.83134952,
```

```
            "profit":0.9988726119663824,
            "usd_x_exchange":"binance",
            "usd_x_value":1.40318698,
            "x_btc_exchange":"kraken",
            "x_btc_value":3.948e-5
         }
  }
   }
}
```


Como se muestra en el ejemplo JSON, las primeras claves pertenecen a los distintos tipos de estrategias que se disponen en el sistema. Dentro de cada una de ellas están todos los mercados que se están monitorizando, y a su vez, dentro de cada uno de los mercados, todas las combinaciones pertinentes entre *exchanges* con la información de los valores de mercados en ellos y el beneficio.

Teniendo toda la información calculada ya disponible, solo hace falta un medio para servir esta información al cliente. Se ha tomado la decisión de montar un servidor Web para que, a través de una API Rest, publicar esta información en un formato JSON.

El servidor Web también se ejecuta sobre Elixir y se ha configurado para que funcione en el puerto 8080, utilizando las tecnologías Cowboy y Plug. Cowboy es un pequeño servidor HTTP rápido y que provee tolerancia a fallos. Plug consiste en una utilidad que proporciona una especificación para componentes de aplicaciones web y adaptadores para servidores web (sería el equivalente a Rack en Ruby o Express en Elixir).

La forma en que estas utilidades funciona facilita la tarea de definir servicios y su funcionamiento. Para obtener la información que se calcula en el módulo *ArbitrageCalculator* se creó un endpoint GET /values. Cuando se llama a este servicio, el Servidor Web se intenta comunicar con el módulo *ArbitrageCalculator* para que le proporcione la estructura de datos actualizada con todas las oportunidades de arbitraje disponibles en ese instante. Cuando recibe la estructura con todos los datos, responde al cliente con código 200 y la estructura en JSON, que antes de poder servirse en JSON debe transformarse mediante la librería Jason puesto que Elixir no soporta JSON de forma nativa.

Como detalle adicional, la aplicación cuenta con un fichero de configuración donde se establecen algunas variables de entorno para el funcionamiento de la aplicación. Entre estas variables, están las credenciales para la base de datos MySQL (host, usuario, contraseña, base de datos) y una variable que determina si la aplicación está en modo depuración (*enable/disable*). Si el modo Debug está activado, durante la ejecución aparecerán logs para visualizar así la ejecución del sistema. Como el log de mensajes provoca algunos retrasos, se recomienda pararlo cuando el sistema está en producción.

3.3.6. Funcionalidad adicional: histórico de beneficios

Aprovechando toda la información que se calcula como parte de la funcionalidad principal, se ha tomado la decisión de añadir otras funcionalidades extra para otorgar

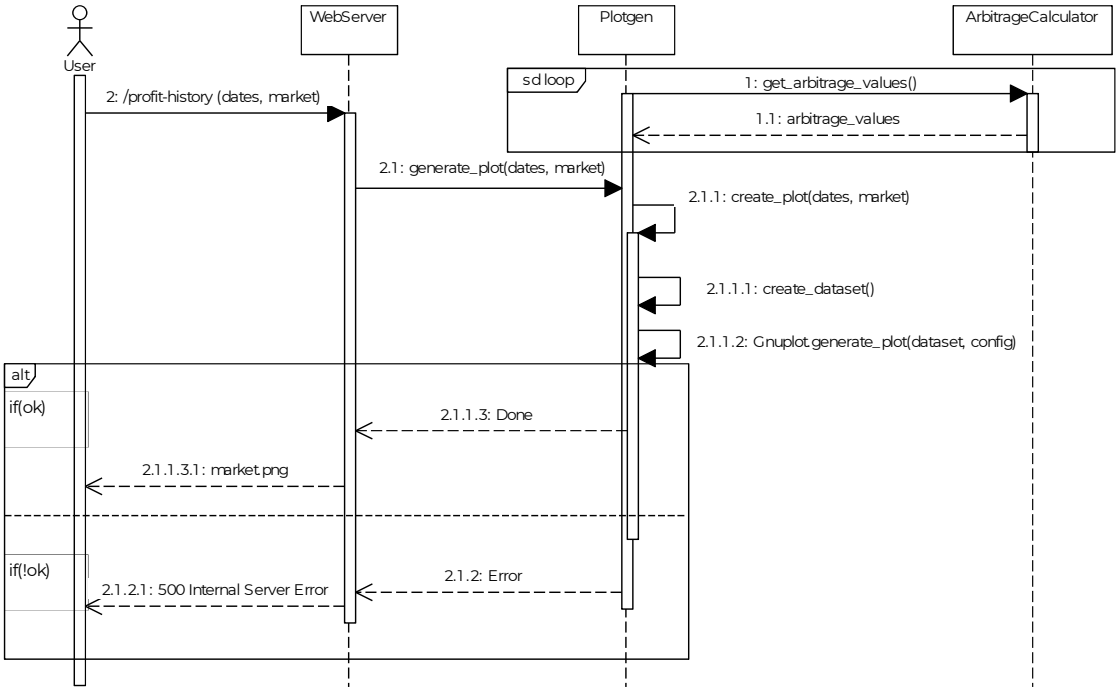

Figura 3.7: Diagrama de secuencia para histórico de beneficios

más información al usuario.

Por una parte, se tomó la decisión de usuario mostrar gráficamente un histórico de los beneficios que han ido otorgando las oportunidades de arbitraje que se han ido calculando. La idea radica en que el usuario pueda solicitar generar un gráfico, en un intervalo de tiempo determinado y un mercado en concreto (dado por el usuario). El gráfico mostraría el beneficio otorgado (eje Y) en función del tiempo (eje X), pudiéndose observar de esta forma la evolución de los beneficios. Además, el gráfico tendrá una línea por cada par de *exchanges* para comparar también los beneficios entre varios pares de *exchanges*.

La figura 3.7 muestra el funcionamiento de esta característica añadida. Para esta función se ha creado un módulo aparte dedicado exclusivamente a esta funcionalidad, llamado *Plotgen*. Este módulo cuenta con dos subhilos. Uno de ellos se encarga de ir creando el histórico de datos. Su misión es enviar solicitudes periódicas al módulo *ArbitrageCalculator* (cada 5 minutos) y de esta forma, creará su estructura de datos donde va a guardar todos los históricos de beneficios por cada par de *exchanges*. El hilo principal, por su parte, va a encargarse de generar los gráficos bajo demanda del usuario. Obtendrá como parámetros el mercado del que desea ver el histórico y el intervalo de tiempo (fecha y hora tanto de inicio como de fin).

Para la creación de gráficos automáticamente se utiliza la herramienta de software libre Gnuplot. Para comunicarse desde Elixir hacia esta utilidad de una forma sencilla, se ha utilizado la interfaz conocida como Gnuplot Elixir [17]. Se trata de una interfaz

Figura 3.8: Ejemplo de gráfico generado por el sistema

que usa los puertos de Erlang para lograr comunicarse con Gnuplot, haciendo compatible también su estructura de datos para generar los gráficos.

Para generar un gráfico se necesita tanto el conjunto de datos que se desean representar como los parámetros de configuración. El conjunto de datos está representado por una matriz, siendo cada vector una línea en el gráfico. Cada elemento vector es una tupla, donde el primer elemento el punto en el eje X y el segundo elemento el punto del eje Y. En este caso concreto, el primer elemento será una fecha/hora y el segundo elemento será el beneficio en ese instante. Respecto a los parámetros de configuración, se otorga el nombre del gráfico (en este caso, el nombre del mercado), el formato en que se va a generar el gráfico y la resolución (por defecto, en formato PNG, sobre el mismo directorio, con el nombre *Mercado*.PNG con una resolución de 1920x1080). También se establece que el eje de las abscisas tenga formato horario en vez de formato numérico, y finalmente se indican todas las líneas que va a ver en el gráfico con sus respectivos nombres, de tal forma que el gráfico pueda mostrar una pequeña leyenda indicando a qué par de *exchanges* pertenece cada una de las líneas.

La figura 3.8 muestra un ejemplo del resultado de estos gráficos. En este caso particular se ha solicitado que se genere un gráfico sobre el mercado BTC/USD desde el 14 de Junio de 2021 a las 13:00h (UTC) hasta las 14:25 del mismo día. En el gráfico aparecen tres líneas correspondientes a los cruces entre Binance-Bitfinex, Binance-Kraken y Bitfinex-Kraken. El gráfico muestra dónde se están dando mejores oportunidades de arbitraje a lo largo del tiempo. En este caso concreto parece apreciarse que el cruce Binance-Bitfinex está otorgando mejores oportunidades de arbitraje.

Para hacer accesible al cliente la generación de gráficos bajo demanda se ha habilitado en el servidor web un nuevo endpoint. En este endpoint el usuario tendrá que pasar los parámetros (tipo GET), entre los que se incluyen el mercado, la fecha de inicio, el tiempo de inicio, la fecha de inicio y la fecha de fin. El sistema convertirá las fechas y horas a un formato `DateTime` para que se pueda operar con facilidad. Se comunicará con el módulo *Plotgen* para solicitar la generación de un gráfico. Cuando se genera el gráfico, aparece una nueva imagen en el sistema que es servida directamente al cliente. En caso de que el cliente haya insertado fechas entre las que no hay datos, el sistema devolverá una imagen vacía.

3.3.7. Funcionalidad adicional: soportes y resistencias

Aprovechando que el sistema mantiene una estructura de datos con los valores de varios *exchanges*, es interesante también dar la oportunidad al usuario de que haga un estudio en el mercado mediante el estudio de soportes y resistencias.

Los soportes y las resistencias son parte fundamental de los análisis de los mercados financieros. Concretamente, se tratan de aquellos puntos donde el precio se detiene y continúa con un comportamiento contrario al que estaba haciendo.

- El soporte es el punto en el que el precio frena su descenso y comienza a subir.
- La resistencia es el punto en el que el precio frena su subida para comenzar a bajar.

Los traders tienden a comprar y vender en estos puntos, eligiendo los mejores niveles. Cuantas más veces en el tiempo el precio tiene a repetir un determinado comportamiento en el mismo nivel, más sencillo es predecir el movimiento del precio en el futuro. Existen varios tipos de soportes y resistencias, pero se focaliza la atención en los de tipo fijo. Los niveles fijos hacen referencia a zonas de soporte y resistencia que no se pueden modificar: seguirán siendo válidos mientras el precio no rompa (cruce) a través de ellos.

La figura 3.9 muestra cómo es la interacción de esta nueva funcionalidad. Para su implementación, se ha habilitado un módulo llamado *SupportResistance*. Al igual que el anterior, también utiliza dos subprocesos. Uno de ellos se encarga de enviar peticiones periódicas al módulo *ArbitrageCalculator* para conocer el último valor de todos los mercados de cada *exchange* y guardarlo en una estructura de datos en forma de histórico de valores. Por su parte, el hilo principal es el que se va a encargar de generar el gráfico bajo la demanda del usuario.

Para generar un gráfico con soportes y resistencias, el usuario tiene que otorgar el mercado y el *exchange* sobre el que se desea realizar el estudio, y también los intervalos de fecha y hora sobre los que se debe realizar el estudio (igual que para la funcionalidad anterior).

Cuando un usuario solicita crear un gráfico, el sistema tomará el histórico de datos para el mercado y el *exchange* solicitado y rango horario solicitado, y buscará los soportes y las resistencias. Para ello, un algoritmo recorrerá todos los puntos comparándolo con sus vecinos. Para un determinado punto n, si los valores de sus dos vecinos en la izquierda

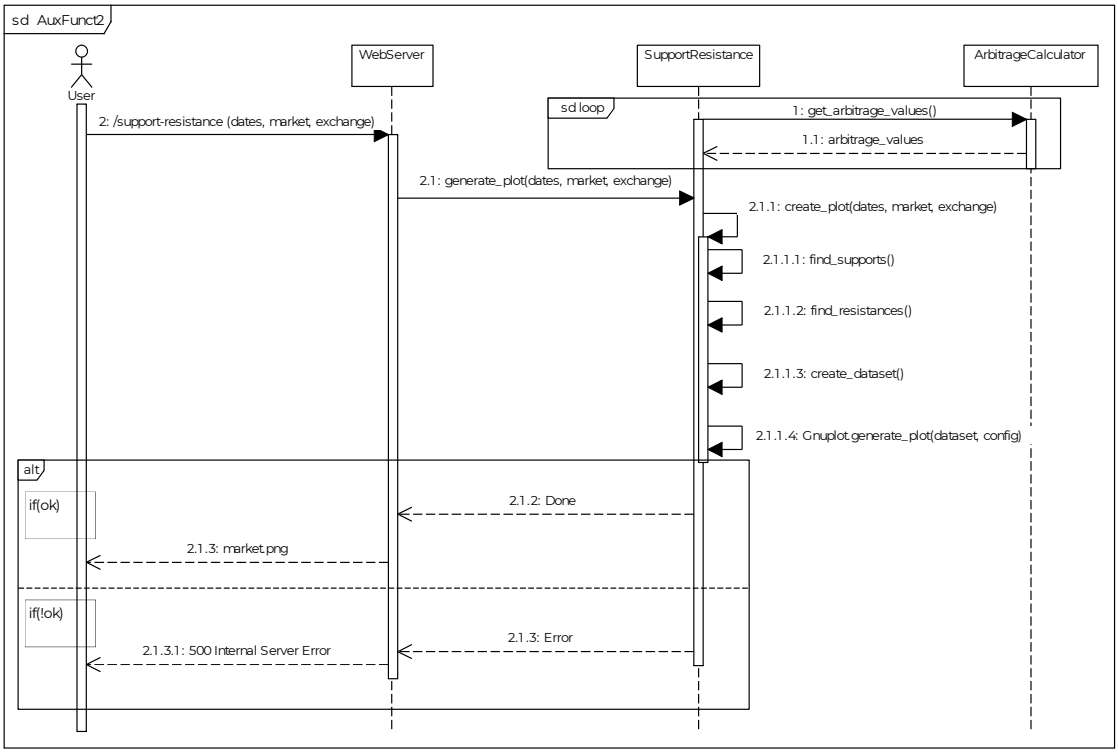

Figura 3.9: Diagrama de secuencia para funcionalidad Soportes/Resistencias

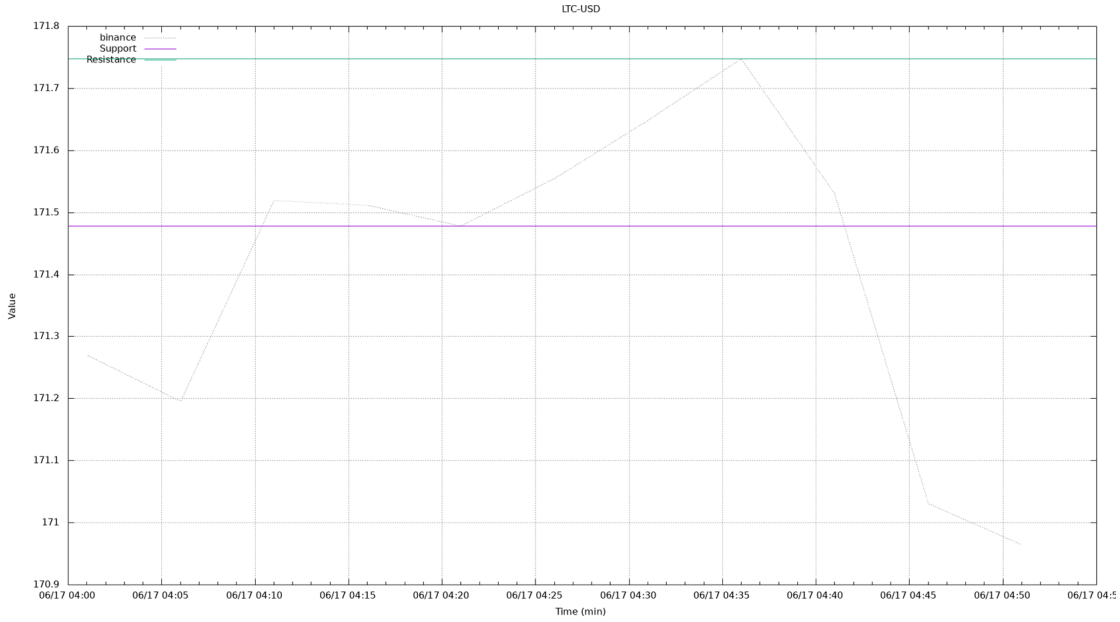

Figura 3.10: Ejemplo de gráfico generado por el sistema

$v(n-1), v(n-2)$ y los valores de sus dos puntos vecinos $v(n+1), v(n+2)$, son todos menores que n, y además se cumple que $v(n-1) > v(n-2)$ y $v(n+1) > v(n+2)$, se tiene una resistencia. El cálculo de soportes es muy parecido, solo que en este caso, en vez de ser el mayor debe ser el menor de sus vecinos, cumpliéndose por tanto que $v(n-1) < v(n-2)$ y $v(n+1) < v(n+2)$. Cuando se encuentra un soporte y una resistencia, quedan guardados. Para la representación gráfica de nuevo se recurre a Gnuplot. Se muestra un ejemplo de la gráfica obtenida en la figura 3.10. Como se puede apreciar en la imagen, aparece la evolución de los valores del mercado junto con las líneas horizontales, indicando los soportes y las resistencias encontradas.

Para facilitar al usuario la generación de estas gráficas, al igual que en la anterior funcionalidad, se ha añadido un nuevo endpoint en el servidor Web. Este endpoint solicita los mismos datos que la funcionalidad anterior, pero además también se solicita el *exchange* sobre el que se desea realizar el estudio.

3.3.8. BOT de Telegram

Por último, se describe la parte de la arquitectura más visible de cara al usuario. Un BOT de Telegram se trata de un programa que se comporta como un chat normal, en la que la que, en vez de interactuar con una persona, se interactúa con un software que atiende los mensajes y proporciona respuestas. Así, Telegram proporciona una interfaz que hace que la interacción con el sistema por parte del usuario sea sencilla.

A pesar de la ventaja de evitar tener que implementar una aplicación adicional es-

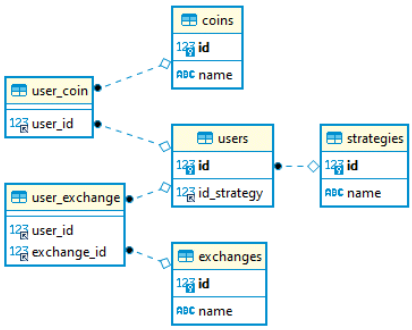

Figura 3.11: Diagrama Entidad-Relación de la base de datos utilizada para el Bot.

pecífica para este propósito, cabe destacar sus posibles desventajas: la infraestructura no está bajo control propio y el diseño se limita a un formato de tipo mensaje, sin la posibilidad de desarrollar mucho más.

Existen también otras limitaciones respecto a la comunicación con Telegram. Considerando que los *exchanges* tienen una actualización cada unos 100ms aproximadamente, en caso de que el sistema detecte una rentabilidad entre dos *exchanges*, no sería lógico enviar un mensaje cada 100ms, en primer lugar porque Telegram no lo permitiría (límite de mensajes) y en segundo lugar porque se saturaría el cliente ante tantos mensajes.

Para este trabajo, se ha realizado un BOT en Telegram para que el usuario pueda recibir notificaciones periódicas acerca de las oportunidades de arbitraje que hay disponibles. Se le da la posibilidad al usuario de suscribirse al BOT y que pueda configurar sus preferencias: sobre qué mercados desea recibir notificaciones, sobre qué *exchange*s, y mediante qué estrategia de arbitraje de las disponibles.

Como ya se mencionó en apartados anteriores, se ha utilizado Python ya que así se puede demostrar el desacoplamiento del sistema y además facilita el desarrollo debido a la existencia de varias librerías para trabajar con Telegram de una forma sencilla.

Para guardar la configuración de cada usuario que se escriba, se requiere una base de datos. Se utilizará la base de datos MySQL para guardar sus preferencias. En la figura 3.11 se muestra un diagrama Entidad-Relación, donde se muestran todas las tablas de la base de datos y sus relaciones. En la tabla *users* se guardan todos los usuarios que se han suscrito al BOT, junto con el ID de la estrategia de cálculo que han seleccionado. Todas las estrategias disponibles se guardan en una tabla llamada *strategies*. También hay tablas para guardar los *exchange*s y los mercados disponibles en el sistema. Al ser relaciones N – N con los usuarios, hará falta también una tabla para relacionar los *exchanges* con los usuarios y los mercados con los usuarios.

La implementación del BOT realizada se basa sobre dos hilos. El hilo principal del programa se encarga de las peticiones por parte del usuario y el segundo hilo se encarga de enviar las oportunidades de arbitraje. Más concretamente, este hilo realiza peticiones

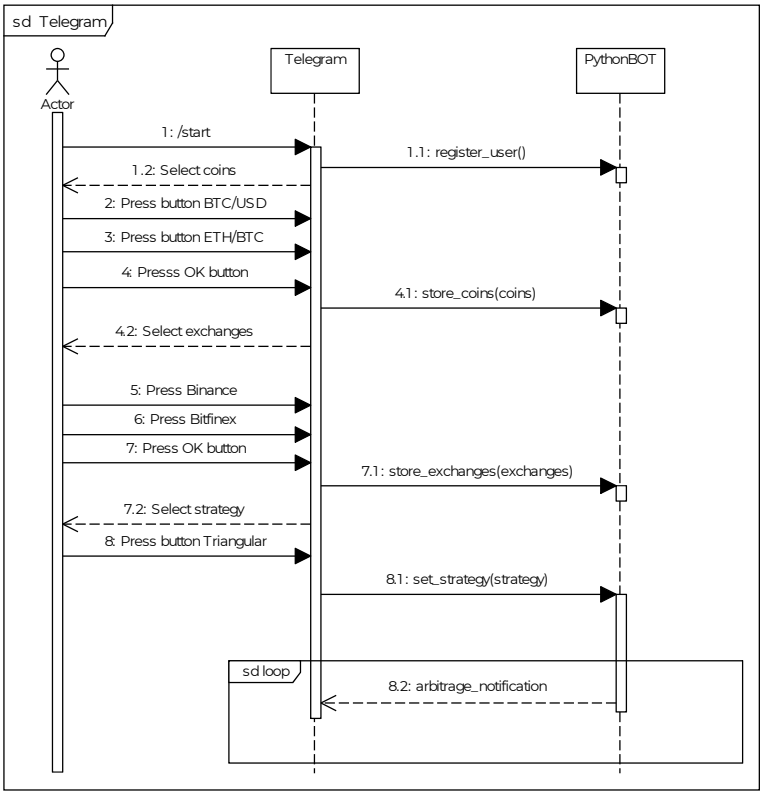

Figura 3.12: Diagrama de secuencia para la interacción con el BOT.

periódicas al endpoint del servidor Web en Elixir para obtener todas las oportunidades de arbitraje e informar a todos los usuarios, aplicando las preferencias de cada uno de ellos.

Como se aprecia en la figura 3.12, la interacción comienza cuando el usuario envía el comando `/start` al BOT. Éste le responderá con una botonera, pudiendo seleccionar todos los mercados de los que desea recibir notificaciones de oportunidades de arbitraje. Cuando confirma con el botón OK, el BOT le solicitará que seleccione los *exchanges* de los que desea ser notificado, y tras pulsar el botón OK nuevamente, dará a elegir la estrategia de arbitraje de la que desea ser informado. Habiendo completado la configuración, el usuario comenzará notificaciones con las oportunidades de arbitraje cada cierto periodo de tiempo. Las figuras 3.13 y 3.14 muestran la interfaz del propio Bot y la forma en que se interactúa.

El usuario puede solicitar también creación de gráficas sobre el histórico de beneficios. Así, mediante el comando `/history <fecha inicio> <fecha fin> <mercado>` el BOT otorgará el gráfico mostrando el histórico de los beneficios para el mercado y los rangos horarios dados por el usuario.

Por último, el bot también permite obtener los soportes y las resistencias de un merca-

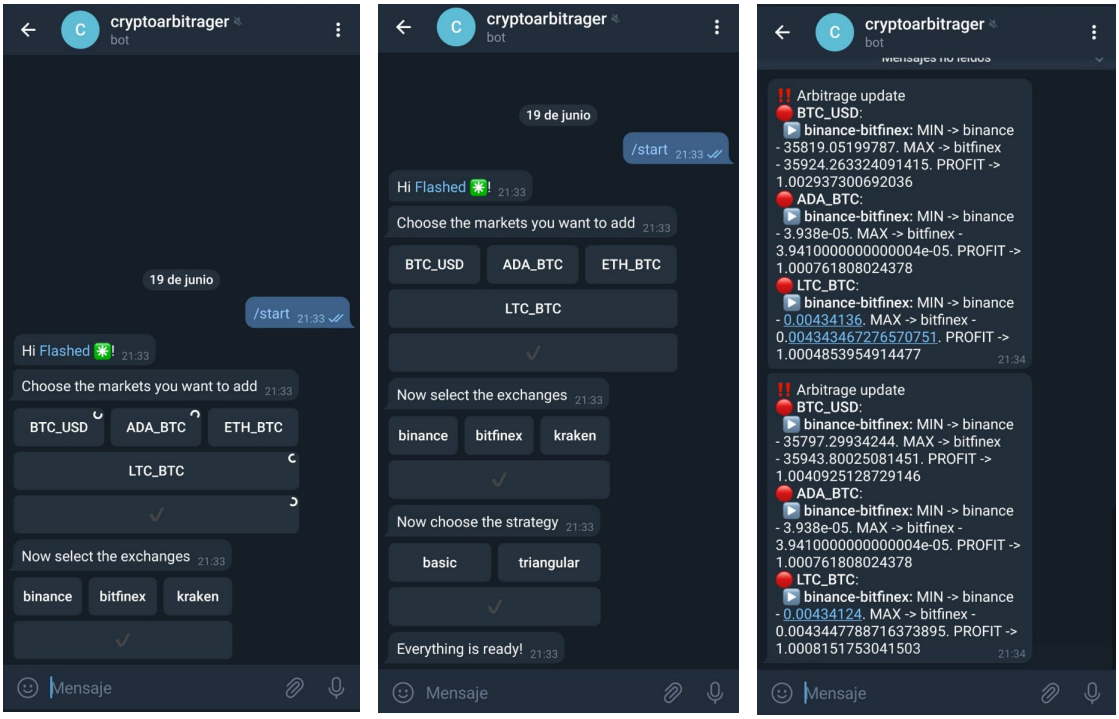

Figura 3.13: Capturas de la funcionalidad principal en Telegram

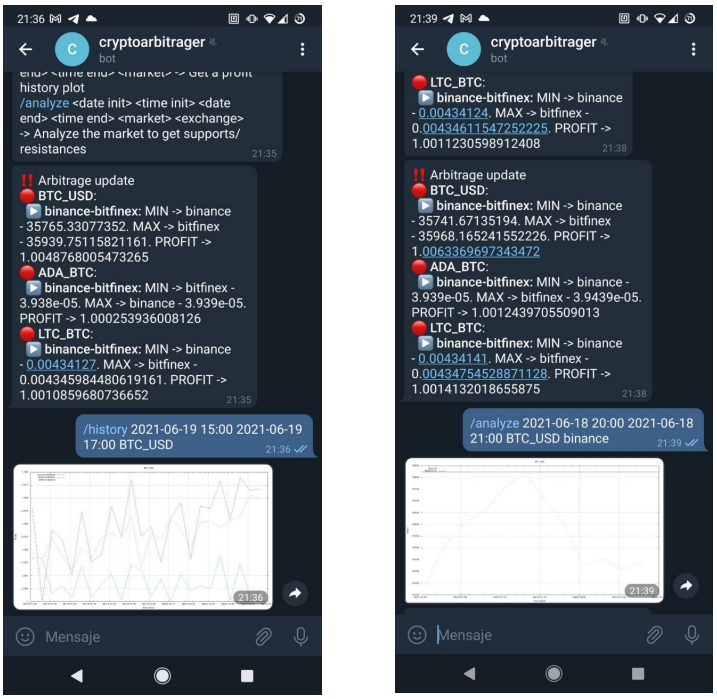

Figura 3.14: Capturas de las funcionalidades adicionales en Telegram

do para un rango de tiempo determinado, de forma similar que la anterior. El usuario puede aplicar el comando `/analyze <fecha inicio> <fecha fin> <mercado> <exchange>` y recibirá el gráfico mostrando los soportes y las resistencias encontrados.

3.4. Despliegue

Tal y como se muestra en la figura 3.15 se ha tomado la decisión de desplegar todo el sistema en tres máquinas distintas. La primera de ellas, SQL Server, se trata de una máquina Windows Server 2012, con 1 vCore y 2 GB de RAM de la empresa Hostigger, que mantiene activo un servidor MySQL 8.0, con las bases de datos para la infraestructura principal y para el BOT de Telegram. Se ha tomado esta decisión debido a que ya se disponía de este servidor con anterioridad, listo para funcionar sin la necesidad de tener que instalar nada. Para desplegar las bases de datos desde cero tan solo es necesaria la instalación del servidor MySQL e importar dos scripts SQL.

El BOT de Telegram, por su parte, se ha decidido desplegarlo en un servidor Debian 8.3, con 4 Cores y 12 GB de RAM, de la Universidad de Zaragoza. Para desplegarlo es necesario tener Python instalado. Primero es necesario instalar algunas dependen-

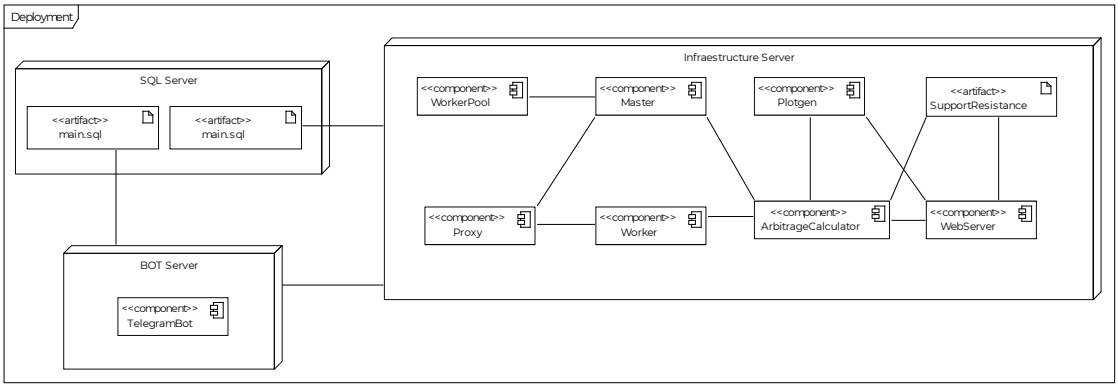

Figura 3.15: Diagrama de despliegue

cias, como son *telegram-python* o *mysql-python*. A través del gestor de dependencias *pip* puede hacerse fácilmente. También habrá que prestar atención al fichero .env, donde se encuentran algunas variables de configuración, como son la API Key para el BOT de Telegram y las credenciales para la base de datos MySQL. Habiendo tenido en cuenta lo anteriormente mencionado, el programa podrá lanzarse sin ninguna complicación.

Por último, y el más importante de todos, es la infraestructura. Se ha decidido desplegarla en un servidor Ubuntu Server 20.04, con 6 vCores y 6 GB de RAM, de la empresa Hostigger. Para el despliegue será necesario tener instalo Erlang y Elixir. Además habrá que instalar algunas dependencias. Por cuestiones de arquitectura y desacoplamiento, el servidor web se encuentra en un proyecto aparte, por lo que habrá que repetir estos pasos también para el servidor web.

A pesar de que se podría haber desplegado la infraestructura entre varios nodos distintos, dado que su diseño distribuido así lo permite, se tomó la decisión de desplegarlo todo en una misma máquina, pues la carga de trabajo que debe realizar en el estado actual puede ser soportada perfectamente por la máquina seleccionada. En la figura 3.16 se muestra la ejecución de la infraestructura ya desplegado.

Todo lo desarrollado para este trabajo, se ha publicado en un repositorio en GitHub, disponible en `https://github.com/Kifixo/distributed-si-crypto-arbitrage` bajo licencia GNU/GPLv3.

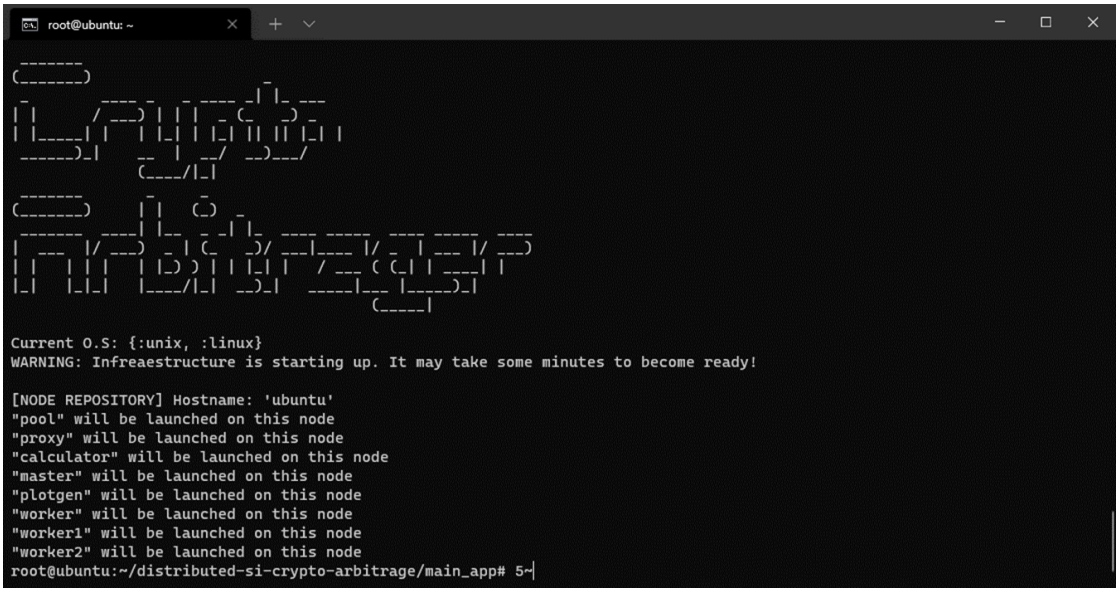

Figura 3.16: Sistema en ejecución

Capítulo 4

Problemas encontrados

El desarrollo de este trabajo ha sido largo y complicado. Como se iba a trabajar sobre conceptos y tecnologías sobre los que no se tenía mucha experiencia, han aparecido ciertas dificultades que se han ido solventando con éxito durante el desarrollo del proyecto.

La falta de experiencia iba a reflejarse sobre la tecnología que se ha utilizado para casi todo el desarrollo: Elixir. Se trata de un lenguaje con un paradigma funcional, con el que apenas se había trabajado anteriormente. El hecho de ser un lenguaje moderno no ha ayudado, pues tampoco existe mucha documentación a la recurrir cuando hay algún problema. No obstante, cabe recalcar que la documentación oficial es bastante completa y algunos problemas podían solucionarse con una lectura rápida sobre esta documentación.

Otro problema importante ha sido la implementación de un diseño realizado sobre un diagrama de clases. Elixir no es un lenguaje orientado a objetos, por lo que plasmar el diseño realizado en una tecnología de este tipo ha sido complicado. Pese a ello, se descubrió que el lenguaje posee algunos mecanismos que permite obtener resultados similares. Por ejemplo, las interfaces pueden implementarse mediante el uso de Behaviours.

El diseño arquitectural tampoco era sencillo, por lo que había que tener cuidado a la hora de implementar puesto que una mala implementación podía suponer que a la hora de agregar al sistema nuevas estrategias, nuevos mercados o nuevos exchanges, hubiera que reescribir muchas partes del código.

Tampoco se contaba con experiencia en la generación de gráficos automáticamente. Para este trabajo, como ya se ha comentado, se ha usado Gnuplot, del cual se partía con nula experiencia. El principal problema encontrado fue que se usaba una interfaz para Elixir que permitía comunicarse con Gnuplot para generar los gráficos. Al tratarse de una interfaz no oficial bastante reciente, era susceptible a fallos. Además, en caso de fallo al generar el gráfico la interfaz no otorgaba ningún tipo de información adicional sobre el error que indicara donde está el fallo. Todo esto llevó a realizar algún pequeño cambio dentro de la propia librería de la interfaz por fallos de implementación, que pese a estar marcados como incidencia en su repositorio oficial de Github, no habían sido corregidos [17].

Por último, en fase de despliegue se observó que algunos módulos comenzaban a fallar provocando su caída al poco tiempo de haber lanzado la infraestructura. Esto era causado

debido al exceso de logs que eran utilizados para depuración. Se llegó a la conclusión de que un volumen tal elevado de mensajes provocaba desbordamiento de buffer de salida, por lo que se tomó la decisión de crear un "Modo debug". En fase de despliegue no han habido más fallos, todo el sistema ha podido estar lanzado y operando durante semanas con total normalidad y sin ninguna anomalía.

Capítulo 5

Aplicaciones similares

La idea de la aplicación que se ha desarrollado no es ninguna novedad. Las aplicaciones para arbitraje automático comenzaron a existir desde que comenzaron a aparecer varios *exchanges*. De hecho, algunos de estas aplicaciones se han tomado como referencia previo comienzo del trabajo para observar cómo trabajaban y tomar algunas ideas.

A continuación, se mencionan algunas aplicaciones que se han tomado como referencia.

- Crypto Arbitrage Framework [18]: se trata de un framework implementado en Python con ccxt (una librería que facilita el trading de criptomonedas) y cplex (una librería para la resolución de problemas de programación lineal). Se trata de una de las aplicaciones más completas. Posee una estrategia de cálculo multicamino, es decir, una estrategia triangular extendida que tiene en cuenta N mercados, aplicando estrategias de programación lineal y optimización para encontrar el mejor camino de forma rápida. No obstante, pese a presumir de escalabilidad, es un aspecto que se debería mejorar, logrando una solución como la que se ha desarrollado en la aplicación de este trabajo.

- Binance Triangle Arbitrage [19]: se trata de otra propuesta para monitorizar oportunidades de arbitraje, pero está tan solo centrado en un solo *exchange* (Binance) y mediante la aplicación de una estrategia triangular, que no sigue ningún algoritmo de optimización, si no que se realiza mediante fuerza bruta. No cuenta además con ningún mecanismo de escalabilidad.

- bitcoin-arbitrage - opportunity detector and automated trading [20]: es un sistema implementado en Python que tiene soporte para más de 10 exchanges distintos, incluso tiene soporte para operaciones automáticas entre dos *exchanges*. No obstante, solo busca oportunidades de arbitraje mediante una estrategia básica y no incluye ningún mecanismo para esalabilidad.

- Peregrine [21]: se describe como una librería en Python para realizar arbitraje entre más de 120 exchanges, obteniendo hasta 38.000 pares. Se trata por tanto de un programa bastante completo. La utilidad permite búsqueda de oportunidades de arbitraje mediante la estrategia multicamino.

- Bitcoin Arbitrage Trading Bot [22]: se trata de otra utilidad desarrollada en Python para este mismo fin. Utiliza la estrategia básica con alguna pequeña variante. Tiene la característica de que posee una interfaz web para visualizar todas las oportunidades que van apareciendo.
- Crypto Arbitrage [23]: se trata de una pequeña utilidad implementada en C#, que soporta 8 Exchanges, pero tan solo un mercado (LTC/BTC). Utiliza la estrategia básica para calcular las oportunidades y mostrarlas por consola. La aplicación, lejos de ser diseñada para ser utilizada realmente, parece una prueba de concepto para comprobar el comportamiento en cuanto a arbitraje de criptomonedas.

Algunas aplicaciones presentadas aquí presentan características muy ambiciosas. Sin embargo, a nivel general se aprecia una ausencia de factores que se consideran importantes para aplicaciones de este tipo, como son la escalabilidad o su diseño arquitectural (muchas de ellas son simples pruebas de concepto, sin visión a futuro). Mientras que las aplicaciones comentadas presentan limitaciones respecto a estas características, el sistema desarrollado destaca sobre ellas aportándolas.

Capítulo 6

Conclusiones

Realizar este trabajo ha otorgado la oportunidad de adquirir grandes conocimientos sobre el mundo de las criptomonedas y los mercados financieros. También se ha logrado gran experiencia con el paradigma de programación funcional y con tecnologías concretas como lenguajes de programación Erlang/Elixir y Python, con los cuales apenas se había trabajado hasta el comienzo desde este trabajo. También se ha aprendido a controlar generadores de gráficos como son Gnuplot, lo cual puede ser bastante útil para el desarrollo de proyectos futuros.

Con todo el tiempo y esfuerzo invertido en este trabajo, se ha logrado desarrollar toda una infraestructura con una arquitectura óptima para su extensibilidad y algunas características que la hacen única, como el factor de la escalabilidad. No obstante, el objetivo principal con el que se desarrolló este sistema pudo no verse saciado: poder ganar dinero asumiendo riesgo prácticamente nulo. El sistema no hace operaciones de forma automatizada, lo que implica que el usuario deberá hacer las operaciones de forma manual, y hay que ser muy rápido para hacer estas operaciones. Los beneficios se garantizan si un beneficio entre pares de *exchanges* perdura en el tiempo, pero si es puntual será complicado obtener algún beneficio tras las operaciones.

A pesar de que se ha conseguido desarrollar un sistema con bastante potencial, no deja de ser susceptible a algunas mejoras. La primera mejora que sería interesante aplicar es la opción de añadir mecanismos de compra automática. De esta forma sería más sencillo sacar provecho de las oportunidades de arbitraje que tiene el sistema. Esto por su parte también tendría una desventaja, y es que se estaría confiando el propio capital a un sistema que puede fallar si no se desarrolla con cuidado, lo que podría suponer pérdidas de dinero considerables.

Otro aspecto de mejora es que la aplicación no cuenta con tests automatizados. Para el desarrollo se han realizado tests de forma manual, cubriendo tanto casos de éxito como casos de fallo para verificar la correcta funcionalidad y la tolerancia a fallos. No obstante, sería interesante la implementación de tests automatizados para que cada vez que se realizan cambios en alguna parte del sistema, se pueda verificar de una forma sencilla que todo el conjunto sigue funcionando de forma correcta. Además, esto no sería una tarea complicada, pues Elixir posee un módulo (llamado ExUnit) a través del cual se

pueden implementar tests de unidad de una forma sencilla.

Por último, también habría sido interesante añadir más *exchanges* al sistema para que a su vez pudieran calcular sobre más mercados y de esta forma obtener más oportunidades de arbitraje. No obstante, la arquitectura está diseñada para integrar nuevos *exchanges* de forma sencilla y muy trivial con un mínimo desarrollo.

Bibliografía

[1] Bitcoin21. Bitcoin prehistory - reddit, 2019. [En línea; Consulta 1 de Junio de 2021] Disponible en: https://www.reddit.com/r/Bitcoin/comments/amdc0s/bitcoin_prehistory/.

[2] Stressgrid. Benchmarking Go vs Node vs Elixir, January 2020. [En línea; Consulta 13 de Junio de 2021] Disponible en: https://stressgrid.com/blog/benchmarking_go_vs_node_vs_elixir/.

[3] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System., 2008. [En línea; Consulta 1 de Junio de 2021] Formato PDF. Disponible en: https://bitcoin.org/bitcoin.pdf.

[4] David Chaum. Blind Signatures for Untraceable Payments, 1983. [En línea; Consulta 1 de Junio de 2021] Formato PDF. Disponible en: https://sceweb.sce.uhcl.edu/yang/teaching/csci5234WebSecurityFall2011/Chaum-blind-signatures.PDF.

[5] Cynthia Dwork and Moni Naor. *Pricing via Processing or Combatting Junk Mail.* IBM Almaden Research Center, 1993.

[6] Adam Back. Hashcash - A Denial of Service Counter-Measure, August 2002. [En línea; Consulta 3 de Junio de 2021] Formato PDF. Disponible en: http://www.hashcash.org/hashcash.pdf.

[7] Wei Dai. B-Money, 1998. [En línea; Consulta 4 de Junio de 2021] Formato TXT. Disponible en: http://www.weidai.com/bmoney.txt.

[8] Robert McMillan. The Inside Story of Mt. Gox, Bitcoin's 460 Million Dollar Disaster, March 2014. [En línea; Consulta 4 de Junio de 2021]. Disponible en: https://www.wired.com/2014/03/bitcoin-exchange/.

[9] Bloomberg. How to Steal 500 Million Dollar in Cryptocurrency, January 2018. [En línea; Consulta 13 de Junio de 2021] Disponible en: https://fortune.com/2018/01/31/coincheck-hack-how/.

[10] AIT News. Blockchain Hackers Stole 3.8 Billion Dollars in 122 Attacks in 2020, January 2021. [En línea; Consulta 13 de Junio de 2021] Disponible en: https://aithority.com/technology/blockchain/blockchain-hackers-stole-3-8-billion-in-122-attacks-in-2020/.

[11] Yago Montero Castellanos. Arbitraje financiero, February 2014. [En línea; Consulta 7 de Junio de 2021]. Disponible en: https://economipedia.com/definiciones/arbitraje.html.

[12] Hans Erik Eriksson. *Bussines Modeling with UML*. John Wiley and Sons Inc, 2000.

[13] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. *Head First Design Patterns*. O'Reilly Media, Inc., October 2004.

[14] Public Rest API for Binance. [En línea; Consulta 14 de Junio de 2021] Disponible en: https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md.

[15] Bitfinex API Documentation. [En línea; Consulta 14 de Junio de 2021] Disponible en: https://docs.bitfinex.com/docs/introduction.

[16] Kraken REST API. [En línea; Consulta 14 de Junio de 2021] Disponible en: https://docs.kraken.com/rest/.

[17] devstopfix. Gnuplot elixir - github repository. [En línea; Consulta 27 de Mayo de 2021] Disponible en: https://github.com/devstopfix/gnuplot-elixir.

[18] hzjkn. Crypto Arbitrage Framework - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/hzjkn/crypto-arbitrage-framework.

[19] bmino. Binance Triangle Arbitrage - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/bmino/binance-triangle-arbitrage.

[20] maxme. bitcoin-arbitrage - opportunity detector and automated trading - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/maxme/bitcoin-arbitrage.

[21] wardbradt. Peregrine - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/wardbradt/peregrine.

[22] mammuth. Bitcoin Arbitrage Trading Bot - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/mammuth/bitcoin-arbitrage-trading-bot.

[23] dougdellolio. Crypto Arbitrage - Github Repository. [En línea; Consulta 18 de Junio de 2021] Disponible en: https://github.com/dougdellolio/crypto-arbitrage-csharp.

Apéndice A

Horas de Trabajo

En la figura A.1 se muestra el tiempo invertido en cada uno de los tipos de tarea. En total se ha trabajado un total de 324,5 horas entre 75 tareas. La Figura A.2 contiene un diagrama de Gantt que ilustra cómo se ha planificado este trabajo. Como se puede apreciar, comienza con un periodo de investigación, donde se adquieren conocimiento acerca del tema que se está tratando. Poco después, comienza una fase de diseño en la cual se estudia la arquitectura del sistema a desarrollar para a continuación, comenzar con una fase de desarrollo donde se llevarán a cabo todos los objetivos. Esta se convierte en la más larga del sistema. Finalmente, queda la elaboración de la memoria, aunque se ha utilizado trabajo redactado durante el desarrollo de las otras fases.

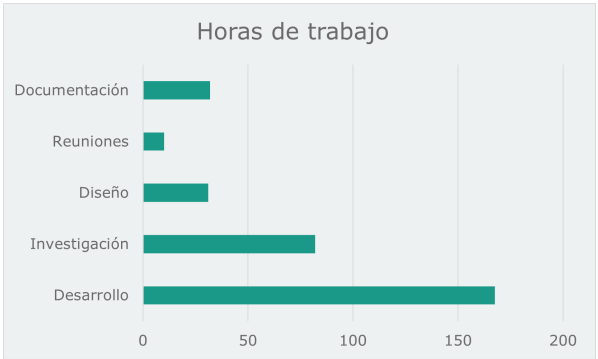

Figura A.1: Desglose de horas empleadas por tarea.

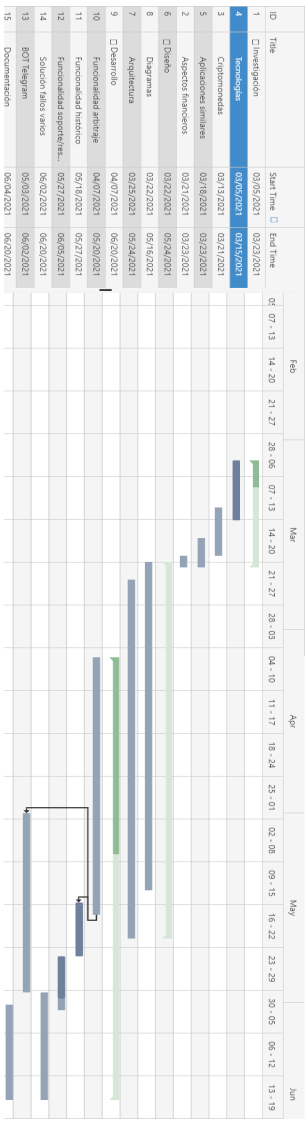

Figura A.2: Diagrama de Gantt.