ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERIA DE COMPUTADORES

PROTOTIPO DE SISTEMA DE RECOLECCIÓN DE DATOS Y ANÁLISIS PARA LA TOMA DE DECISIONES EN SISTEMAS DE BACKUP

SYSTEM PROTOTYPE FOR DATA GATHERING AND ANALYSIS FOR DECISION MAKING APPLIED TO BACKUP SYSTEMS

Realizado por
Leonardo Scovenna Ramírez
Tutorizado por
Guillermo Pérez Trabado
Departamento
Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Junio de 2017

Fecha defensa:
El Secretario del Tribunal

Resumen

Actualmente la mayoría de software de copia de seguridad está orientado a medianas y grandes empresas. Puesto que éstas cuentan con suficientes recursos personales y materiales, para la industria no ha sido necesario invertir en evolucionar los paradigmas clásicos de copias de seguridad ya que el beneficio que tendría dentro de su marco de negocio no es representativo. Esta situación provoca que los profesionales independientes y PYMES no reciban legado tecnológico aplicable a sus necesidades, de forma directa. Para utilizar las mismas tecnologías sería necesario un desembolso significativo en hardware y personal suficientemente cualificado. En este trabajo se estudiará mediante la implementación de un prototipo la viabilidad de un sistema de monitorización de las acciones del usuario en el sistema de ficheros, almacenamiento y posterior análisis de las mismas. Este sistema tendrá como objetivo recopilar datos de uso de los ficheros, aplicar métodos estadísticos para ayudar a la detección de la estructura definida por el uso de los datos de usuario, clasificar los ficheros utilizados según criterios de relevancia, y como resultado práctico, reducir la complejidad de la toma de decisiones relacionada con las copias de seguridad.

Palabras claves:

Copias de seguridad, Sistema de ficheros, Análisis de datos, Patrón de uso de datos.

Abstract

Most backup software is currently focused on medium-sized and large companies. Since these companies have enough economic muscle to dedicate both personnel and resources to disaster recovery, the industry has focused its business in selling high end backup infrastructures. Few development has been done to evolve the classic backup paradigm as the profit they would obtain within their business framework is unrepresentative. As a result, independent professionals and SMEs do not receive any technological heritage applicable to their needs in a straightforward manner. The result in usually unsatisfactory as the technology requires plenty of resources, if brute force is used, or skilled personnel. This paper will study the feasibility of a monitoring system prototype related to the user´s actions in the file system and its subsequent analysis. The aim of this system is to gather data on file usage, apply statistical methods to help detection of structure defined by access to user files, classify the files used according to relevance criteria, and consequently, reduce the complexity of decision making concerning backup files.

Keywords:

Backups, File system, Data analysis, Prototype, Data usage pattern.

Índice

1. Introducción

1.1. Motivación

En la actualidad y en el contexto de las copias de seguridad existen diferentes paradigmas. Cabe destacar que son paradigmas clásicos, bien definidos y rara vez modificados. Uno de los más utilizados es el modelo de copia de seguridad incremental que consiste en una copia de seguridad completa seguida de una serie de copias dependientes de la primera, que solo guardan los cambios realizados en el intervalo de tiempo que las separa. Este modelo es ampliamente utilizado puesto que consigue un compromiso entre el volumen de información generado y la frecuencia con la que se pueden realizar las copias. Cabe destacar que uno de los problemas más difíciles de resolver en éste área es determinar la cantidad de riesgo asumible por fallo de los sistemas para ajustar la frecuencia de las copias de seguridad y que el volumen de datos generado sea compatible con el espacio de almacenamiento disponible, velocidad de E/S, etc. A fin de cuentas, nos encontramos con un problema de ingeniería donde los costes son uno de los factores limitantes y muchas veces el más importante.

Para la configuración de sistemas de copia de seguridad existen a grandes rasgos dos modelos:

1. Una configuración para guardar imágenes del sistema completo
2. Una configuración para guardar árboles de directorios.

Si analizamos la primera opción vemos que estaremos guardando ficheros de sistema constantemente y por norma general de poco valor para el usuario/empresa.

Atendiendo a la segunda opción podremos ver que la poca flexibilidad de la configuración no tiene en cuenta los cambios naturales que un usuario puede necesitar a lo largo del tiempo, dejando muchas veces ficheros importantes sin respaldar. A esta situación hay que añadir que la forma clásica de control de cambios en un fichero se realiza a través de la fecha de modificación ya que su tamaño no es un indicador fiable [1]. En condiciones óptimas debería hacerse una comparación de contenido entre la copia almacenada y el fichero actual pero este procedimiento

genera un elevado volumen de E/S [2] y computación que hace del método ineficiente y por tanto menos deseable.

En los últimos años los esfuerzos de investigación y desarrollo se han centrado en la mejora de los algoritmos de compresión y deduplicación [3, 4, 5] de la información a salvaguardar, sin preocuparse por el filtrado en origen. Por tanto, la única manera de restringir el volumen es que el administrador de sistemas tenga un conocimiento de la finalidad de cada directorio del espacio de ficheros y que manualmente incluya aquellos de gran valor y excluya los de poco valor en términos de impacto en el negocio (Business Impact Analysis) independientemente de la frecuencia con la que sean actualizados los ficheros.

En lo que respecta a las PYMES se presenta una problemática específica. Almacenar grandes volúmenes de datos es altamente costoso a nivel de infraestructura y componentes HW haciéndolo inviable para este tipo de negocio. Almacenar sólo algunas carpetas de forma estricta es viable para empresas grandes a medianas, con administradores a tiempo completo y trabajadores con posiciones fijas y generalmente acotadas. En el caso de las PYMES, los perfiles de usuario se difuminan rápidamente y sus necesidades económicas impiden contratar un administrador a tiempo completo. Como resultado podemos encontrar una cantidad pequeña de ordenadores con toda la información crítica para el funcionamiento de la empresa y poca o nula gestión de las copias de seguridad.

Hay que destacar que la cantidad de tiempo (normalmente meses) que un administrador a tiempo parcial tiene que invertir antes de conocer el funcionamiento real de una PYME, así como sus perfiles de usuario y la estructura de información generada.

Por tanto, la idea fundamental del proyecto es explorar la vía de una reducción del volumen de las copias incrementales mediante un análisis automático de la importancia real de los ficheros para el negocio basado en una monitorización continúa de la actividad de los ordenadores a medio plazo y unos procesos de análisis más o menos inteligentes que determinen el valor de cada directorio para la empresa.

1.2. Objetivos

El objetivo del proyecto es generar un prototipo de la herramienta antes mencionada, de tal manera que se pudiesen recoger los datos de uso referente al sistema de ficheros de un ordenador con Windows, enviarlos a una base de datos donde se almacenarán y analizarán devolviendo a medio y largo plazo inferencias interesantes para los técnicos y/o trabajadores de la PYME.

La recogida de datos se hará a través de un programa en lenguaje a determinar dependiendo de la conveniencia de las librerías ofrecidas por Windows a tal efecto.

La información recolectada se almacenará en una base de datos, a determinar por la fase de investigación inicial de este mismo trabajo.

La información se procesará utilizando métodos o heurísticas que permitan hacer inferencias sobre los datos obtenidos.

Por último, la información se presentará al usuario en un formato fácilmente entendible independientemente de la formación o conocimientos del mismo.

1.3. Estado del arte

En esta parte del trabajo exploraremos el estado del arte actual en temas de copias de seguridad. Comenzaremos con una breve introducción sobre la teoría necesaria para pasar a exponer las diferentes soluciones ofrecidas.

1.3.1. Copias de seguridad

Una copia de seguridad es una copia de los datos de un sistema, normalmente almacenada fuera de dicho sistema y que nos da la posibilidad de recuperar los datos si algo malo le sucediera al conjunto de datos originales. Generalmente los datos que se suelen respaldar son aquellos que se consideran importantes en términos de impacto de negocio, ejemplos de esto puede ser la libreta de contactos de clientes, la contabilidad, los proyectos actuales y pasados, el historial de los clientes o cualquier otro fichero relevante.

Para hacer las copias de seguridad existen varios modelos extendidos que se utilizan con normalidad y casi en exclusividad por los softwares de copias de seguridad. Los

modelos más utilizados son: La copia de seguridad completa, la copia de seguridad incremental y la copia de seguridad diferencial. Además, existen otros modelos como el Abuelo-Padre-Hijo o el de las torres de Hanoi.

- **Copia de seguridad completa:**

El modelo de copia de seguridad completa es el más simple de todos. Cada vez que se realiza una copia de seguridad completa se guarda una copia completa de todos los ficheros que se han configurado para respaldar. Este tipo de copia de seguridad es muy efectivo en términos de simpleza pues no hay posibilidad para el error.

- **Copia de seguridad incremental:**

El modelo de copia de seguridad incremental sólo copia los datos que han variado desde la última operación de backup de cualquier tipo. Se suele utilizar la fecha y hora de última modificación para detectar qué ficheros son necesarios copiar. Las copias de seguridad incrementales necesitan menos espacio y tiempo cada vez que se realizan, por el contrario, para ser recuperadas necesitan de todas las copias incrementales anteriores hasta llegar al backup completo más próximo. Es decir, si alguno de los elementos de la cadena se corrompe, solo se podrá recuperar hasta el anterior al corrompido.

- **Copia de seguridad diferencial:**

El modelo de copia de seguridad diferencial copia todos los datos que han cambiado desde el último backup completo realizado. Cada vez que se hace genera una diferencia entre el estado actual del sistema y el estado en el que estaba cuando se hizo la primera copia. Es más robusto que el incremental, pero tiene el problema que necesita más espacio.

- **Modelos híbridos:**

Generalmente los modelos que se utilizan son una combinación entre los tres anteriormente explicados. Por ejemplo, se puede hacer una copia completa semanal y una copia incremental o diferencial diaria.

Ejemplo de algunos de los sistemas de backup a los que nos referimos en párrafos anteriores son:

Cobian Backup:

Software gratuito de copias de seguridad muy utilizado por pequeñas empresas y particulares. Es sencillo y permite hacer copias completas, diferenciales e incrementales. Carece de flexibilidad a la hora de hacer backups en remoto y no cuenta con herramientas para la recuperación. Además sus creadores ya no prestan soporte. [6]

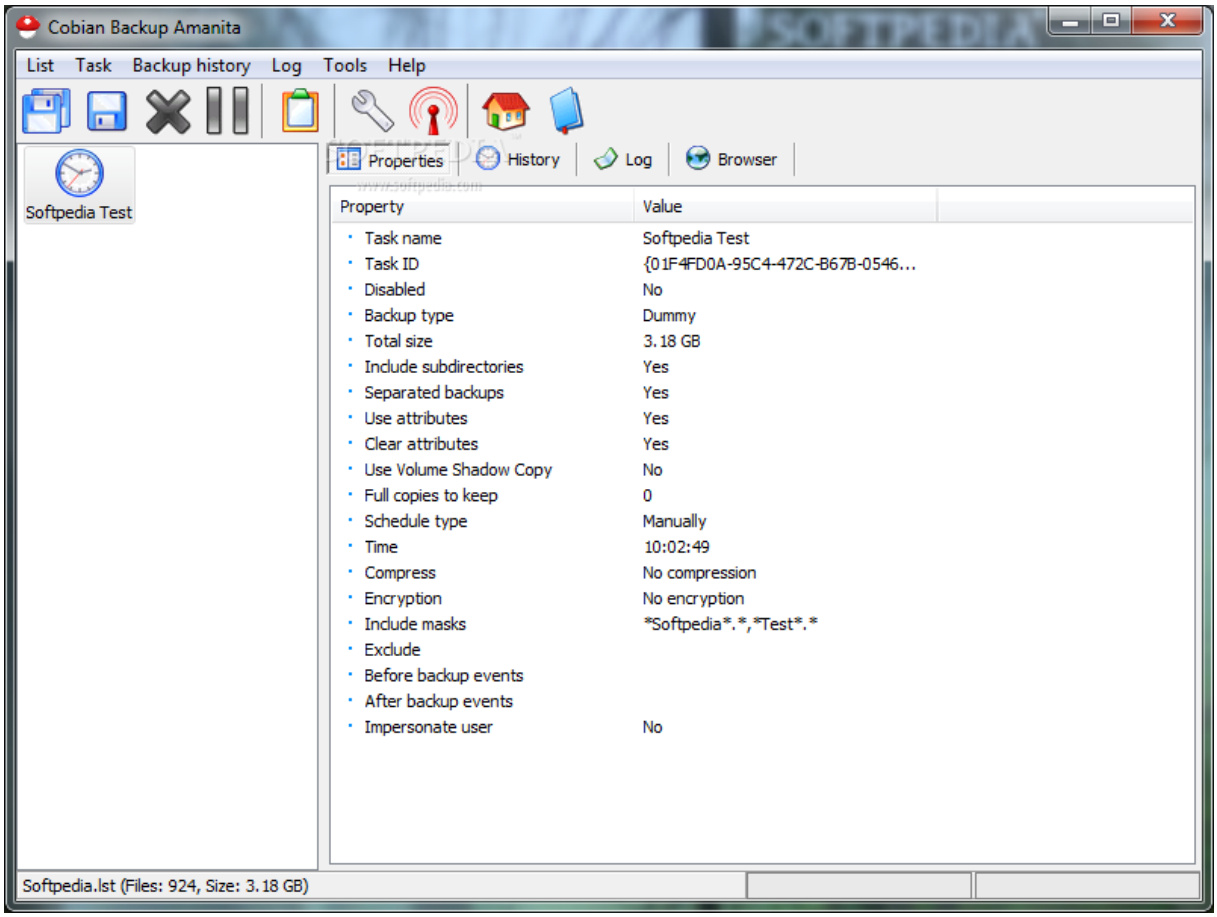

Figura 1.1: Cobian Backup

Acronis Backup:

Software de pago de copias de seguridad orientado principalmente para empresas. Cuenta con diferentes niveles de servicio según licencia. Muy poco flexible en las formas de uso de las licencias adquiridas (una licencia distinta para cada uso). Las versiones más asequibles se encuentran muy restringidas. [7]

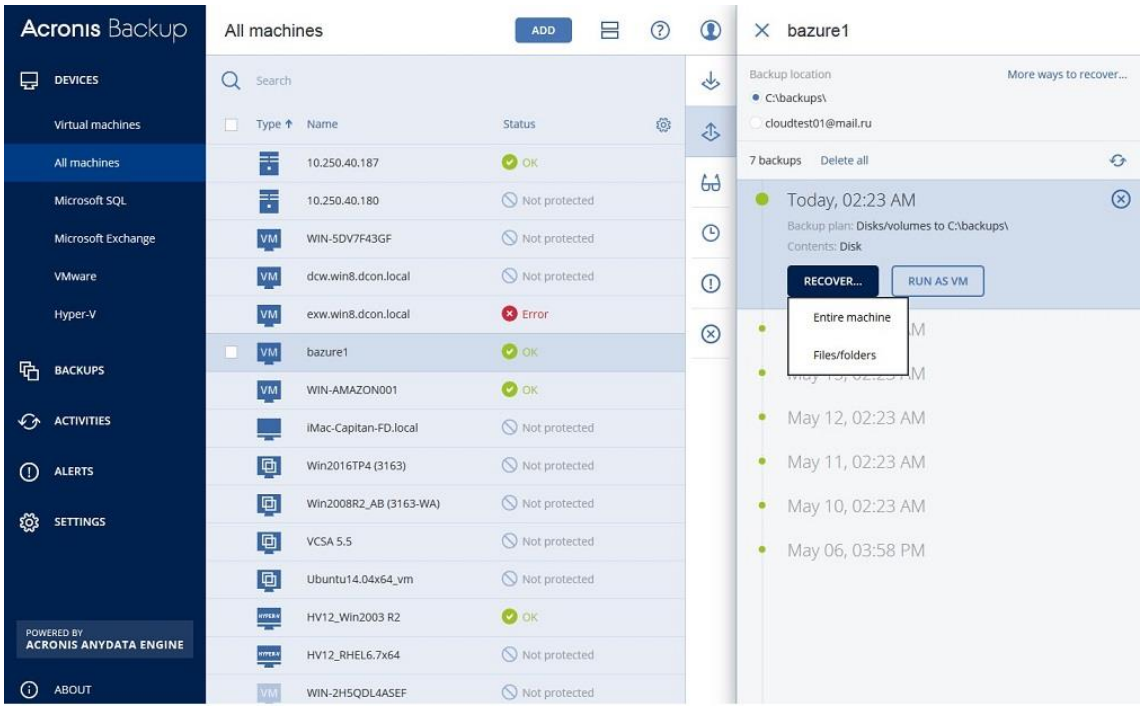

Figura 1.2: Acronis Backup

Bacula:

Software libre empresarial de copias de seguridad. Posiblemente una de las soluciones más completas gratuitas del mercado. Funciona bajo el modelo cliente servidor. Es muy complejo de utilizar y necesita de una administración activa constante. Algunas empresas tienen soluciones basadas en este software elaboradas para que sea más sencillo de utilizar. [8]

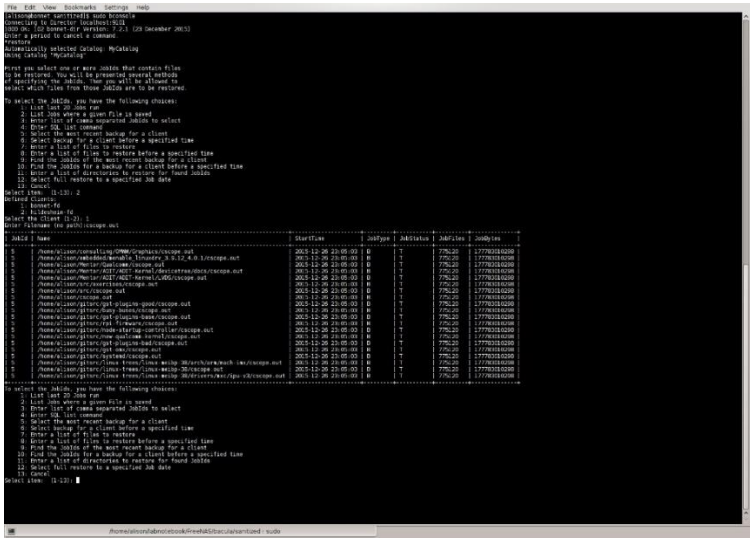

Figura 1.3: Bacula consola

Actualmente en el mercado encontramos diferentes soluciones para aplicar a las PYMES. Unas en extremo simples que no acaban de cumplir todas las necesidades y otras que por el contrario exceden las necesidades, la complejidad y los costos. No hemos encontrado aplicaciones que monitoricen la actividad del sistema de ficheros para tomar decisiones más inteligentes sobre las copias de seguridad.

1.4. Tecnologías utilizadas

Lenguaje C# con Framework .NET:

.NET es un Framework desarrollado por Microsoft que se ejecuta principalmente en Microsoft Windows. Incluye gran cantidad de librerías y proporciona interoperabilidad entre diferentes lenguajes. Los programas escritos para .NET se ejecutan en una capa superior a una máquina virtual que les proporciona servicios de seguridad, manejo de memoria, manejo de excepciones entre otros. Dentro de este Framework encontramos las librerías más importantes para interactuar con Windows en este trabajo. El lenguaje utilizado para la codificación del software cliente es C# por ser un lenguaje de alto nivel, compatible con .NET. [9, 10, 11]

Virtualización:

Para generar los diferentes ordenadores necesarios para desplegar el prototipo se ha utilizado virtualización clásica y virtualización de contenedores. Para mapear las máquinas físicas hemos utilizado tanto VMware Workstation como ESXi mientras que para los nodos de la base de datos se ha optado por utilizar contenedores de Docker para facilitar el proceso de configuración y despliegue. [12, 13]

Apache Cassandra:

Apache Cassandra es una base de datos NoSQL distribuida y basada en un modelo de almacenamiento de «clave-valor», de código abierto que está escrita en Java. Permite grandes volúmenes de datos en forma distribuida. Su objetivo principal es la escalabilidad lineal y la disponibilidad. La arquitectura distribuida de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo P2P con lo que la redundancia es máxima. Está desarrollada por Apache Software Foundation. [14, 15, 16]

R:

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico. Se trata de uno de los lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy popular en el campo de la minería de datos, la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo y gráficas.

Hemos utilizado este lenguaje para el probar manualmente los métodos que más tarde aplicaríamos a los datos de forma automática con otras herramientas. [17]

Python:

Python es un lenguaje de alto nivel de propósito general. Es un lenguaje interpretado. Muchas veces es referido como un lenguaje de prototipado ya que existen gran cantidad de librerías disponibles para él y no es un lenguaje complicado a nivel sintáctico.

Hemos utilizado Python para el análisis de datos por la cantidad de métodos matemáticos disponibles y su flexibilidad para conectar con otros elementos del proyecto como la base de datos. [18, 19, 20]

Python Pandas:

Pandas es una librería de software escrita para el lenguaje de programación Python para la manipulación de datos y análisis. En particular ofrece estructuras y operaciones para la manipulación numérica de tablas y series temporales. Pandas es software libre y se encuentra bajo el tipo de licencia BSD.

Pandas ha sido utilizada para implementar los mismos métodos desarrollados con R y que estos fuesen fácilmente ejecutables bajo el lenguaje Python. [21]

Django:

Django es un Framework web de código libre escrito en Python. Sigue la arquitectura Model-View-Template. Es mantenido por una organización independiente y sin ánimo de lucro.

El objetivo de Django es crear páginas web complejas fácilmente, orientadas a los datos obtenidos de una base de datos.

Django ha permitido integración completa entre el sistema de análisis escrito en Python y el sistema de representación. [22, 23, 24]

Plotly:

Plotly es una librería de código libre de representación de gráficas interactivas compatibles con la web. Nos ha permitido crear gráficas más fáciles de interpretar y más amigables que las gráficas clásicas de librerías como ggplot. [25]

Tecnologías Web:

Para la representación se ha utilizado HTML y CSS para la maquetación web. Se ha hecho uso también de Bootstrap para ayudar al maquetado y en otras partes código Javascript para los elementos interactivos.

1.5. Estructura de la memoria

La presente memoria se ha dividido en cuatro grandes bloques.

En este primer bloque de introducción hemos visto la motivación del trabajo, así como el estado del arte y las tecnologías utilizadas, que han hecho posible su desarrollo.

En el segundo bloque justificaremos las tecnologías utilizadas, y desarrollaremos en profundidad las diferentes partes del proyecto. Poniendo especial énfasis en el cliente software, el almacenamiento y procesado de los datos y la representación de los resultados obtenidos.

En el tercer bloque se presentarán los resultados obtenidos.

Finalmente, en el quinto y último bloque se presentan las conclusiones respecto de los resultados y se plantearan los lineamientos de futuros trabajos.

2. Copias de seguridad adaptativas

Copias de seguridad basadas en modificaciones

En la actualidad se continúa utilizando el modelo clásico de backup. Este modelo es especialmente interesante para las grandes empresas, a quienes los costos asociados a esta práctica son asumibles. Sin embargo, para las pequeñas empresas el modelo clásico deriva en unos costes de almacenamiento y personal cualificado que normalmente excede sus posibilidades.

El funcionamiento del formato clásico se basa en realizar una primera copia de seguridad completa que contiene todos los datos del usuario/sistema. Más tarde se realizan copias de seguridad diferenciales o incrementales que contienen los deltas de información. Tanto las copias de seguridad completas como las copias siguientes se realizan con una periodicidad establecida previamente. Para reducir el espacio necesario y el tiempo de ejecución las copias de seguridad completas se suelen realizar con menor periodicidad que las diferenciales/incrementales.

La problemática que nosotros vemos en este formato radica en que el espacio consumido aumenta de forma exponencial. Las copias de seguridad completas incluyen no solo los datos del “working set” sino también toda la información que no se utiliza o no va a ser reutilizado, es decir, el archivo (Archive) y los ficheros temporales generados con el uso diario.

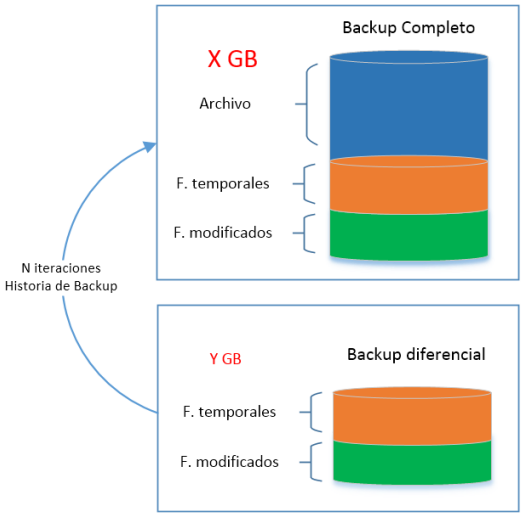

Figura 2.1 Modelo clásico de copia de seguridad

Para evitar, en parte, los problemas enumerados anteriormente existen tres alternativas comunes. La primera de ellas es el aumento de la capacidad de almacenamiento, la segunda el uso de filtros manuales (administrador) y la tercera tecnologías de deduplicación. La compra de medios para aumentar el almacenamiento en condiciones óptimas sólo debería ser necesaria cuando no existan otras alternativas para reducir el espacio requerido. El uso de filtros manuales es posiblemente el método más eficaz para la reducción del espacio, sin embargo, es necesario un conocimiento profundo de los datos almacenados y grandes cantidades de tiempo, que se traducen en unos costos humanos muy elevados. Por último, las técnicas de deduplicación son efectivas para grandes empresas donde existen mucho volumen de datos duplicado, así mismo son costosas a nivel computacional, de almacenamiento temporal y de adquisición de licencias. Precisamente uno de sus principales usos es reducir el almacenamiento necesario mediante la compresión de los datos inmutables repetidos en cada backup completo periódico que se guarda.

Ninguna de las técnicas anteriores es adecuada para las PYMES ya que no cuentan con los mismos recursos que otras empresas más grandes. Por ello sería de interés el desarrollo de una herramienta/método que permita reducir el espacio necesario para las copias de seguridad en este ámbito. La propuesta que nosotros planteamos es el desarrollo de un prototipo de sistema de monitorización y análisis de las actividades de los usuarios, permitiendo una optimización de los recursos disponibles a través del conocimiento generado por la herramienta y la posibilidad de aplicación de técnicas estadísticas/data mining.

Modelo propuesto: copia de seguridad adaptativa

El proyecto gira en torno a una idea central, la combinación de los patrones de actividad del usuario con los metadatos que genera el sistema de ficheros nos posibilita clasificar los ficheros utilizados por categorías como la importancia, el ser o no ser temporales o el pertenecer o no al conjunto de datos “Archivo”, entre otros. Para ello utilizaremos tuplas que representarán eventos en el tiempo, y cada uno de sus elementos representarán una característica específica de dichos eventos. Nos basaremos en el concepto de distancia para los distintos elementos de la tupla con la finalidad de obtener relaciones relevantes entre ellos.

$$(Identificador\ Evento, Fecha\ del\ evento, tipo\ del\ evento, fichero\ asociado, \dots)$$

Figura 2.2: Formato de tupla

Podremos definir por tanto distintas distancias en función de la naturaleza de los datos y estas prepararán a los conjuntos significativos para ser representados y estudiados a posteriori. Un ejemplo de distancia significativo en el sistema de ficheros lo podemos ver cuando ordenamos las distintas carpetas del sistema de forma alfabética y las numeramos. Podemos medir entonces, la distancia entre dos ficheros cualesquiera.

En nuestro proyecto contaremos con una primera fase de recogida de datos. Para ella nos valdremos de un programa que se ejecutará en el espacio de usuario teniendo acceso a información generado por el Kernel del sistema operativo a través de las librerías de sistema (.NET Framework: FileSystemWatcher, EventWatcher). Para ello hemos considerado dos modelos de funcionamiento. El primero, un sistema de escaneo del estado del sistema de ficheros, el segundo, la recogida de datos en tiempo real. El sistema de escaneo fue descartado porque cuando se aumenta la frecuencia de escaneo aumenta significativamente su coste computacional. Debido a la baja carga de actividad del usuario en el tiempo, una recogida de datos en tiempo real no interfiere con la normal actividad. Siendo especialmente adecuado ya que no se pierden eventos y tampoco se incurre en un coste computacional elevado.

Monitorizaremos los eventos del sistema que se han determinado como relevantes. Una vez recogidos los datos serán enviados a la base de datos.

En una segunda fase “Servidor” tendremos los datos disponibles para su consulta. Aplicaremos técnicas de limpieza y preparación de estos datos para que se puedan representar de forma que sean significativos. Aquí haremos uso de las diferentes medidas de distancia que hemos mencionado anteriormente.

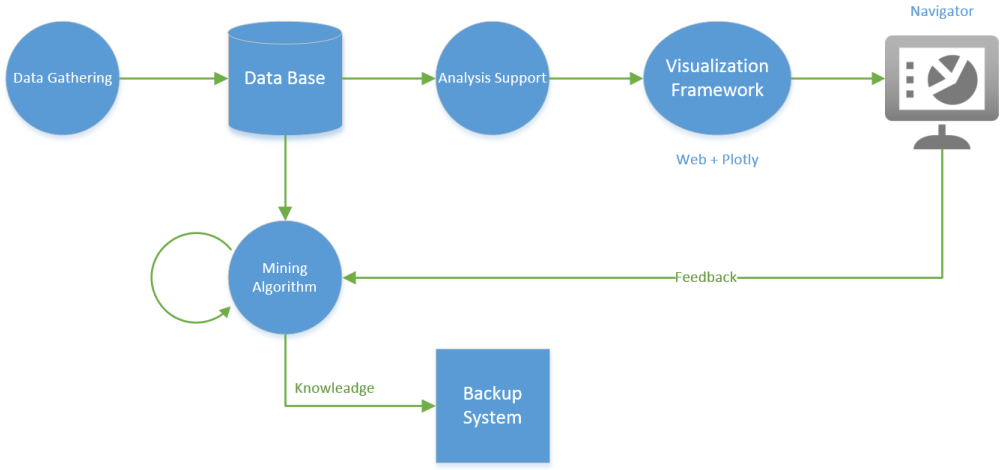

Figura 2.3: Modelo adaptativo de copia de seguridad e infraestructura de análisis

Una vez realizado lo anterior enviaremos los datos al apartado de visualización. Esto nos permitirá observar de forma sencilla los datos para encontrar patrones de interés. Este proceso tendrá como resultado un mayor conocimiento del espacio del problema y por ende de las soluciones aplicables.

En esta fase habrá dos posibles caminos, el primero consistiría en la aplicación manual por parte de un usuario de los conocimientos obtenidos a su sistema de backup. La segunda posibilidad es un estudio en profundidad de los patrones generados para poder generalizarlos de tal forma que puedan ser aplicadas técnicas automáticas para la clasificación y toma de decisiones. El sistema que nosotros plateamos resultaría el punto de partida para la búsqueda dirigida de estos algoritmos.

Finalmente, el resultado sería un conjunto de conocimientos y/o reglas aplicables al sistema de backup con la intención de optimizar los recursos necesarios.

En el sistema de niveles de backup la copia de seguridad completa es representada por el nivel cero y las copias incrementales/diferenciales por niveles superiores. En cada nivel n no se incluye aquello que ya esté incluido en el nivel n-1. Por ello proponemos incluir a esta escala un nivel n = -1 de tal forma que en él se encuentren los ficheros que pertenezcan al archivo. Con este modelo evitaremos hacer copias de seguridad completas que contengan ficheros que no cambian o no son utilizados a lo largo de un período de tiempo determinado. De esta forma obtendríamos la reducción de espacio deseada, sin la necesidad de determinar manualmente que ficheros pertenecen a la clase archivo y cuáles no.

3. Desarrollo

3.1. Introducción

Para la realización de este trabajo hemos tenido que desarrollar un software destinado a la recogida de datos (cliente), basado en las librerías FileSystem Watcher y EventLog de .Net.

Los datos recogidos son almacenados en una base de datos Apache Cassandra, que será posible escalar siempre que sea necesario aumentar la cantidad de datos a gestionar.

Los datos almacenados serán post-procesados utilizando métodos de limpieza, selección y estadística descriptiva. Estos métodos se han codificados en Python con Pandas.

Cuando los datos están listos para su representación son enviados al motor del servidor web, que generará la vista necesaria para el usuario. Para esta fase hemos utilizado Django en combinación con la librería de gráficas Plotly.

3.2. Estructura del proyecto

El proyecto está desarrollado bajo el modelo cliente-servidor. Contaremos con clientes que, mediante nuestro software de recogida de datos, nos proporcionarán la información que a posteriori será almacenada, analizada y representada en el servidor.

Teniendo en cuenta lo anterior, podemos dividir el servidor en tres partes atendiendo a las tareas que realiza. Una primera parte de almacenamiento de datos, una segunda parte con la lógica de análisis y digestión y una tercera dedicada a la representación de los resultados.

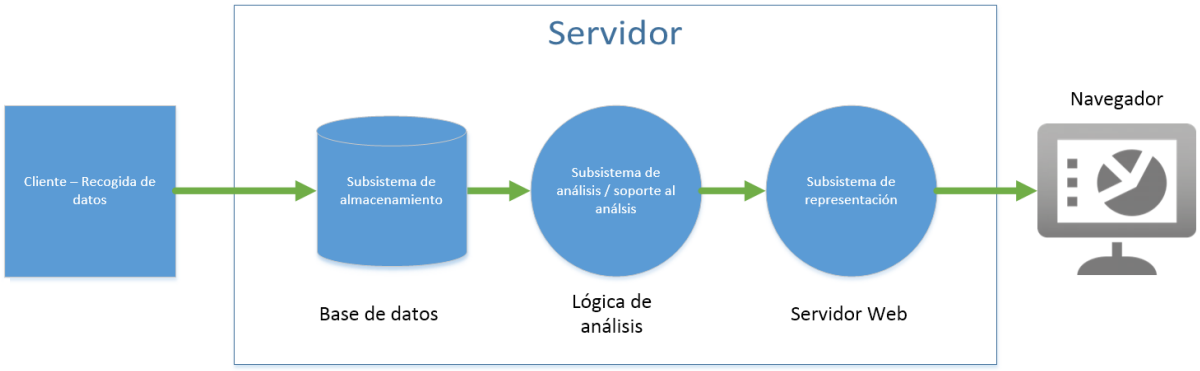

Figura 3.1: Estructura del Servidor

Cliente:

Para este trabajo se ha considerado “cliente” a nuestro software de recogida de datos, que será ejecutado en los ordenadores de los que se obtendrán los datos.

Servidor:

Hemos considerado servidor al conjunto de subsistemas siguientes:

- **Subsistema de almacenamiento:**
  Se utilizará una base de datos para el almacenamiento, orden y posterior consulta de los datos de los clientes.
- **Subsistema de análisis:**
  Una vez dispongamos de los datos será necesario una fase de preparación, filtrado, análisis y digestión.
- **Subsistema de representación de resultados:**
  Para que los resultados sean fácilmente accesibles para los usuarios, habrá un sistema de representación de los resultados. En este caso utilizaremos un servidor web.

3.3. Cliente

3.3.1. Objetivo del cliente

El objetivo del cliente es la recogida de datos previamente determinados y preparado para el posterior análisis en el servidor. El tipo de datos que serán recogidos son los eventos del sistema de ficheros, producto del uso normal del ordenador. Por ejemplo, cuando el usuario crea un documento nuevo el sistema operativo tendrá que realizar un conjunto de operaciones de menor nivel. Nosotros recogeremos estas operaciones a nivel del sistema de ficheros. Ejemplos de estas son “LastAccess”, “Size” o “LastWrite” que se corresponden con accesos, cambio del tamaño o escrituras en ficheros. Los eventos generados son recogidos en un fichero con un formato específico para que sean enviados al servidor.

3.3.2. Plataforma de desarrollo

Para que el desarrollo del presente trabajo resulte de interés general, hemos decidido hacer la recogida de datos en clientes con sistema operativo Microsoft Windows. Es cierto que esto normalmente representa un reto a nivel de programación más elevado que desarrollarlo para Linux, sin embargo, hemos tenido en cuenta que en la actualidad la mayor parte de las empresas utilizan entorno Windows para los puestos de trabajo.

3.3.3. Medios utilizados

Para el desarrollo y pruebas del cliente se ha utilizado un ordenador con procesador Core2duo a 2.4Ghz, 4Gb de RAM y 1TB de disco duro. S.O Windows 8.1 pro.

Se ha escogido esta configuración en particular sabiendo que el público objetivo suele tener equipos con una media de edad de 5 años y generalmente con poco o nulo mantenimiento.

Para que el cliente funcione adecuadamente en este tipo de configuraciones es imprescindible tener en cuenta técnicas de eficiencia en el rendimiento. Por ello hemos desarrollado este software en especial atención a lo anterior. [26]

3.3.4. Paradigma de recogida de datos

La primera aproximación que se ha tenido en cuenta, ha sido desarrollar un escáner de los ficheros del usuario por el cual cada cierto tiempo se hiciera un barrido en el espacio de usuario para ir obteniendo “fotos” del estado para su posterior comparación. Parece una aproximación muy razonable pues no tiene que estar en ejecución todo el tiempo, se puede programar para que los escaneos se realicen en momentos de poca carga de trabajo y de esta forma pasar desapercibido para el usuario.

Ahora bien, el gran problema que se nos presenta es determinar la periodicidad de los escaneos, para no perder parte de la actividad del usuario, ya que la frecuencia de las acciones del usuario puede variar de forma considerable. Eso nos lleva a tener que disminuir el intervalo entre escaneos perdiendo la propiedad más beneficiosa de este modelo, la eficiencia.

Por todo lo anteriormente descrito, hemos descartado el escáner y hemos buscado una segunda opción que es la recogida de datos en tiempo real. Un programa que se encuentre en segundo plano escuchando los eventos del sistema y los recoja a medida que se van produciendo. Podría pensarse que esta aproximación es menos eficiente que la del escáner, sin embargo, teniendo en cuenta que solo estamos monitorizando el espacio de usuario la cantidad de eventos generados es limitada. Como mucho se producirían pequeñas ráfagas para determinadas operaciones. Empíricamente hemos comprobado que esta aproximación funciona adecuadamente siendo casi imperceptible para el usuario.

Datos a obtener:

Los datos que resultan de interés se pueden dividir en dos grandes grupos:

Metadatos de los ficheros:

Dentro de los metadatos consideramos elementos como el nombre del fichero, la extensión, la ruta, la última fecha de acceso, la última fecha de modificación, la fecha de creación, el tamaño, etc.

Operaciones del sistema de ficheros:

En este apartado podemos considerar las operaciones del usuario como son la creación de un fichero, el renombrado, el borrado y el cambio de tamaño.

Si para cada evento generado por el usuario unimos estos dos grupos de datos podemos obtener una aproximación fiable sobre las acciones que se realizan.

3.3.5. Tecnologías y librerías utilizadas

La tecnología clásica para este tipo de software son los MiniFilterDrivers. Un MiniFilterDriver se coloca en el espacio del kernel e intercepta operaciones dirigidas a los drivers de los periféricos, que en este caso sería el disco duro. De esta forma se consiguen operaciones de muy bajo nivel y un nivel de detalle elevado. Por contrapartida se pierde mucha de la semántica que podemos recoger dentro del especio de usuario y la dificultad de programación es muy elevada haciendo el método no viable para un prototipo.

Por todo lo anterior, para la programación del cliente utilizaremos C# como lenguaje ya que encontraremos muchas librerías de alto nivel que encapsulan funciones de sistema programadas originalmente en C++. Algunas de estas funciones son las mismas que utilizaríamos a bajo nivel en el MiniFilterDriver de forma directa.

El cliente, está basado principalmente en dos librerías de sistema de Windows System.IO y System.Diagnostics.

Las clases principales que hemos utilizado son:

System.IO.FileSystemWatcher:
Permite escuchar las notificaciones de cambio del sistema de archivos y genera eventos cuando cambia un directorio o un archivo de un directorio. [27]

System.Diagnostics.EventLog:
Permite escuchar las notificaciones de cambios en el gestor de eventos de Windows. Con él podremos hacer uso automatizado de herramientas de auditoría de Windows. [28]

Otras clases que se han utilizado:

System.Threading:

System.Threading realmente es un espacio de que proporciona clases e interfaces que habilitan la programación multiproceso además de las clases de sincronización de las actividades del subproceso y acceso a datos El System.Threading espacio de nombres proporciona clases e interfaces que habilitan la programación multiproceso. Además de las clases de sincronización de las actividades del subproceso y acceso a datos (Mutex, Monitor, Interlocked, AutoResetEvent, etc.) [29]

System.Collections:

El espacio de nombres System.Collections contiene interfaces y clases que definen varias colecciones de objetos, como listas, colas, matrices de bits, tablas hash y diccionarios. [30]

System.IO.FileInfo:

Proporciona propiedades y métodos de instancia para crear, copiar, eliminar, mover y abrir archivos. [31]

System.Security.AccessControl.FileSecurity:

Representa el control de acceso y seguridad de auditoría de un archivo. [32]

System.IO.DirectoryInfo:

Expone métodos de instancia para crear, mover y enumerar archivos en directorios y subdirectorios. [33]

System.IO.File:

Proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir un solo archivo. [34]

System.IO.Directory:

Expone métodos estáticos para crear, mover y enumerar archivos en directorios y subdirectorios. [35]

System.Security.Cryptrography:

El System.Security.Cryptography espacio de nombres proporciona servicios criptográficos, incluidas la codificación y descodificación segura de datos, así como muchas otras operaciones, como cálculos hash, generación de números aleatorios y la autenticación de mensajes. [36]

3.3.5.1. Funcionamiento de la clase FileSystemWatcher

Esta clase se encarga de generar un objeto “Watcher”, que se registrará a una serie de eventos relacionados con los cambios en el sistema de ficheros. A continuación, nos referiremos a este tipo de eventos como eventos del sistema de ficheros.

Cada vez que suceda un evento del sistema de fichero se lanzará la rutina de manejo de dicho evento en el objeto Watcher.

A continuación, vemos un ejemplo sencillo de inicialización para un objeto Watcher en donde solo es necesario pasarle como argumento el directorio que va a monitorizar. Luego se filtran la clase de eventos que vamos a escuchar, en el caso del ejemplo eventos relacionados con el nombre del fichero. Asociamos la función manejadora del evento y por último habilitamos la monitorización.

```
FileSystemWatcher watcher = new FileSystemWatcher(path);
watcher.NotifyFilter = NotifyFilters.FileName;
watcher.Changed += new FileSystemEventHandler(OnChanged);
watcher.EnableRaisingEvents = true;
```

Figura 3.2: Inicialización para objeto Watcher

```
private void OnChanged(object sender, FileSystemEventArgs e)
{
   Console.WriteLine("Se ha producido un evento OnChanged");
}
```

Figura 3.3: Rutina manejador de evento

Un objeto Watcher puede monitorizar distintos comportamientos de forma específica a través de los filtros NotifyFilters listados en la Tabla 3.1.

Filtro (NotifyFilter)	Acciones	Acción detectada
Attributes	Changed, Renamed, Created Deleted	Registra cambios en los atributos (Solo lectura, oculto y atributos avanzados)
CreationTime	Changed, Renamed, Created Deleted	Registra la creación de nuevos ficheros
LastAccess	Changed, Renamed, Created Deleted	Registra el acceso a un fichero
LastWrite	Changed, Renamed, Created Deleted	Registra una escritura en el fichero
Size	Changed, Renamed, Created Deleted	Registra cambios en el tamaño del fichero
FileName	Changed, Renamed, Created Deleted	Registra la creación de ficheros el cambio de nombre y el borrado
DirectoryName	Changed, Renamed, Created Deleted	Registra la creación de directorios, el cambio de nombre y el borrado
Security	Changed, Renamed, Created Deleted	Registra cambios en los atributos de seguridad (Pestaña seguridad de las propiedades)

Tabla 3.1: Combinaciones para Watchers

El uso que la documentación oficial de la librería propone para la Clase, está orientado a un solo objeto Watcher. Este objeto Watcher tendrá que registrarse para algún tipo de filtro o para todos ellos a la vez. Puesto que además el Watcher tiene 4 rutinas de manejo para las acciones (Changed, Renamed, Created o Deleted) no se podría determinar exactamente qué acción ha disparado tal evento.

Para poder precisar qué tipo de acción ha lanzado un evento del sistema de ficheros concreto hemos optado por crear un Watcher por cada uno de los filtros disponibles “Attributes”, “Creation time”, etc.

Para que todos los Watchers puedan funcionar sin interferirse, los hemos encapsulado en diferentes hebras puesto que encontramos comportamientos extraños cuando se ejecutaban en un solo hilo compartido. Además, éstos nos permiten hacer uso de herramientas de concurrencia como el AutoResetEvent de tal forma que evitamos tener una espera activa.

3.3.5.2. Funcionamiento de la clase EventLog

Esta clase proporciona interacción con los registros de eventos de Windows. En el proyecto ha sido utilizada para leer los eventos relacionados con la auditoría de ficheros de Windows. Debido a que el comportamiento de FileSystemWatcher para el filtro LastAccess no generaba un evento diferente cada vez que se accedía a un fichero concreto, sino que solo lo generaba la primera vez, no servía para nuestros propósitos. Por lo tanto, hemos tenido que utilizar las herramientas de auditoría de ficheros de Windows que recogen en el gestor de eventos una serie de acciones, entre ellas el acceso a los ficheros.

Para la inicialización de EventLog es necesario el nombre de la categoría bajo la que se encuentran los eventos que queremos escuchar. En el caso de la auditoría de ficheros para Windows estos eventos están bajo la categoría “Security”.

Asignamos la rutina de interrupción de eventos para el tipo de evento `EntryWrittenEventHandler` y por último activamos la escucha.

A continuación, podemos ver un ejemplo sencillo rutina de interrupción.

```
eventlog = new EventLog("Security");
eventlog.EntryWritten += new EntryWrittenEventHandler(OnEntryWritten);
eventlog.EnableRaisingEvents = true;
```

Figura 3.4: Inicialización para objeto EventLog

```
public void OnEntryWritten(object source, EntryWrittenEventArgs entryArgs)
{
        Console.WriteLine("Se ha producido un evento");
}
```

Figura 3.5: Rutina manejador de evento

3.3.6. Estructura del cliente

Luego de algunas consideraciones previas se ha optado por una estructura concurrente de escucha de eventos.

Explicaremos ahora brevemente la relación existente entre las clases principales y luego detallaremos las clases restantes.

Contamos con dos clases principales denominadas Watcher y EventTracker, destinadas a la escucha de eventos. Watcher escuchará los eventos del sistema de ficheros, mientras que Event Tracker escuchará los eventos del gestor de eventos de Windows.

Ahora bien. Contando con siete trabajadores (hebras) Watcher y un trabajador Event Tracker gestionados por la Clase Background Watchers.

Todos ellos interactuarán con la clase FileInfo para obtener datos asociados a los ficheros que generan los eventos y a la clase FileWriter para escribir los resultados a un fichero de salida.

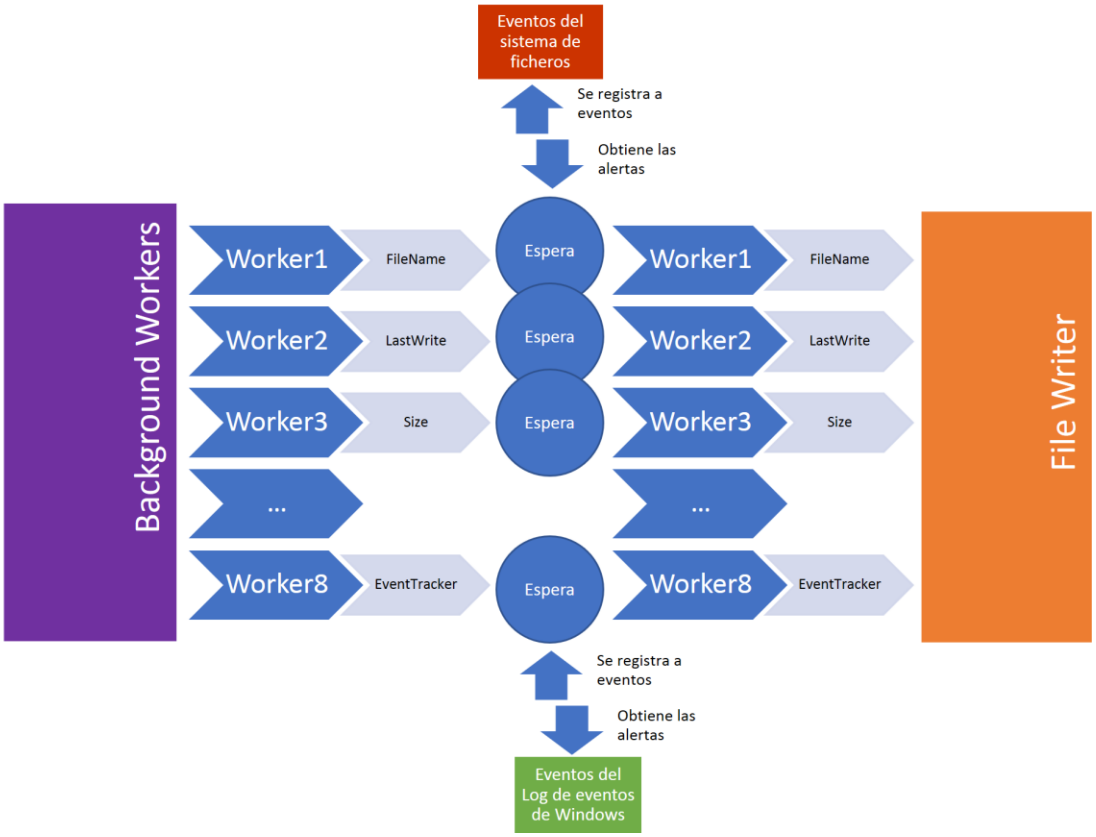

Figura 3.6: Esquema hebras del cliente

Listado detallado de clases:

Program.cs:

Clase principal para inicializar el programa. Recoge los argumentos de inicio como son la ruta que ha de monitorizar y la ruta del fichero de salida.

FileInfo.cs:

Clase que permite recoger información de los ficheros asociados a los eventos. Empaqueta la información para que las otras clases puedan manejarla directamente como cadenas de texto.

TemporalSequence.cs:

Clase que asigna a cada fichero un identificador temporal de uso para poder seguir su actividad en una sesión. Esta clase se ha añadido a posteriori por necesidades de representación gráfica.

Watcher.cs:

Clase envolotorio de FileSystemWatcher para que pueda utilizarse cómodamente en nuestro programa. Contiene la lógica de los manejadores de eventos. Podría decirse que es la clase principal sobre la que está planteado todo el desarrollo.

EventTracker.cs:

Clase envoltorio de EventLog para que pueda utilizarse cómodamente en nuestro programa. Contiene la lógica de los manejadores de eventos. Se ha utilizado tras haber detectado algunas deficiencias puntuales en el funcionamiento de la clase FileSystemWatcher.

BackgroundWorkers.cs:

Clase contenedor para todas las hebras creadas y para la información compartida que utilizan como es el diccionario de identificadores temporales. Contiene la inicialización de las hebras.

FileWriter.cs:

Clase que permite a las hebras escribir en un fichero de salida evitando problemas de concurrencia.

3.3.7. Funcionamiento del cliente

Habiendo explicado la estructura del cliente, pasaremos a detallar el funcionamiento del mismo.

En el momento en que se ejecuta el cliente se crea es una instancia de BackgroundWorkers. Esta instancia de BackgroundWorkers tiene como función preparar a los distintos trabajadores (hebras) y proporcionarles un espacio común para interactuar con otras clases.

En este trabajo se han creado por intermedio de BackgroundWorkers ocho trabajadores (hebras), encargados de escuchar los eventos.

Las 7 primeras hebras corresponden a los filtros de notificaciones (NotifyFilter), para cada uno de los Watchers generados y la última corresponde al encargado de revisar el Gestos de eventos (log de Windows).

Teniendo en cuenta la baja frecuencia de los eventos, utilizar espera activa para cada una de las hebras tendría asociado un costo de potencia de proceso muy elevado.

Para evitar el costo de las esperas activas, hemos optado por utilizar la construcción autoresetevent, que nos permite despertar a la hebra solo cuando es necesario.

En el momento en que se produce un evento, se despierta a la hebra correspondiente para que pueda ejecutar las ordenes específicas para ese evento, una vez finalizadas vuelve al estado inicial de reposo.

Esta vuelta al estado de reposo de cada una de las hebras se produce con la ejecución de waitHandle.WaitOne().

Cuando se genera un evento, se hace una llamada a la rutina de interrupción, en la hebra correspondiente, ésta recoge la información del fichero asociado al evento a través de una instancia de la clase FileInfo y otra de TemporalSequenceHash.

Una vez obtenida la información necesaria, como son el nombre del fichero, la ruta del fichero el tamaño del fichero, etc. Dicha información se concatena y se envía al fichero de salida a través de la clase FileWriter.

3.3.7.1. Sobre la clase Watcher

Watcher es una clase envoltorio para la clase FileSystemWatcher. En ella se ha codificado todo lo necesario para pasar del modelo de procesamiento en serie sugerido por la documentación oficial de la librería, al modelo concurrente propuesto en el apartado de estructura. Además, esta clase contiene código que asigna las acciones adecuadas a cada uno de los Filtros (NotifyFilters). Por último, contiene las rutinas de manejo de interrupciones para cada acción (OnChanged, OnRenamed, OnDeleted, OnCreated).

3.3.7.2. Sobre la clase EventTracker

EventTracker es una clase envoltorio para la clase EventLog. Si atendemos al formato de los trabajadores ejecutados por la clase BackgroudWorkers podemos ver que 7 de ellos son instancias de la clase FileSystemWatcher mientras que el octavo es de la clase EventTracker, esto sucede así por los problemas encontrados para registrar los eventos de acceso a los ficheros.

Inicialmente plantemos que todos los trabajadores pertenecieran a la misma clase, sin embargo, al ejecutar pruebas nos dimos cuenta de que el comportamiento del Watcher para el filtro “LastAccess” no era como cabría esperar. Cuando un fichero era accedido de cualquier manera, ya fuese por el usuario o por el sistema operativo para tareas secundarias se lanzaba un evento “LastAccess”, sin embargo, este fichero parece que quedaba cargado en memoria, impidiendo que se volviesen a lanzar eventos “LastAccess” asociados a él, por largos períodos de tiempo. Entendemos que esto es un mecanismo de eficiencia, del sistema operativo que nos impedía ver los accesos repetidos a un fichero concreto.

3.3.7.3. Sobre la clase FileInfo

La clase FileInfo contiene un conjunto de métodos tanto de Windows como propios, para obtener información sobre los ficheros asociados a los eventos generados. A través de FileInfo se crea un objeto que es capaz de preguntar al sistema sobre el nombre del fichero, su extensión, sus metadatos, etc.

Además, resulta de interés por la particularidad de actuar en las operaciones de borrado como de renombrados, haciendo que estos sean consistentes con el resto de eventos a pesar de contener menos información.

3.3.7.4. Sobre la clase TemporalSequenceHash

La clase TemporalSequenceHash tiene como objetivo asignar un identificador único por ventana temporal. Esta clase ha sido añadida una vez que nos encontrábamos en la fase de análisis de resultados. Hemos visto que necesitábamos un identificador único de fichero en formato numérico para las gráficas de actividad. Para ello hemos utilizado un diccionario concurrente en el que se guardan los hashes de los ficheros utilizados y se le asigna a cada uno un número natural contiguo al anterior.

Para hacer un seguimiento de los ficheros en sistemas de ficheros NTFS existe el atributo *$Object_ID* como identificador único en la vida del fichero. A pesar de parecer muy adecuado para no tener que llevar una estructura auxiliar con los identificadores que nosotros generamos, se presentaron muchas dificultades para utilizarlo con los eventos de borrado.

3.3.7.5. Sobre la clase FileWriter

FileWriter es una clase auxiliar para la escritura de los datos producidos en un fichero de salida. Como la estructura del cliente es concurrente se ha optado por un semáforo como método de sincronización entre hebras ya que la cantidad de eventos producidos en el tiempo no es muy elevada y las hebras pueden permitirse estar encoladas pequeñas fracciones de tiempo. Además, encapsula las órdenes necesarias para hacer una escritura en C# permitiendo claridad en las partes del código donde es necesario.

3.3.8. Formato de los datos

A medida que se van produciendo los eventos del sistema de ficheros o del gestor de eventos de windows, cada una de las hebras recoge la información asociada y la almacena como cadenas de texto.

Los datos generados son guardados en un fichero CSV. Se ha elegido este formato porque permite ser muy flexible en cuanto a la lectura por parte de bases de datos y programas de gestión y análisis de datos en general. Se puede decir que es en este momento el formato estándar para la recolección de datos a tratar con programas como Matlab, RStudio o similares.

Los datos obtenidos y guardados son los siguientes:

Dato	Descripción
Hash del usuario	Hash del identificador único del usuario, utilizado como identificador el e-mail
Fecha del evento	Fechas en la que se ha generado el evento
Watcher	Tipo de NotifyFilter asociado al Watcher que ha generado el evento.
Tipo de evento	Tipo de acción que ha generado el evento.
Hash del fichero	Hash de la ruta del fichero
Ruta	Ruta absoluta del fichero

Nombre	Nombre del fichero sin su ruta
Extensión	Extensión del fichero
Fecha de creación	Fecha de creación del fichero
Fecha de modificación	Fecha de la última modificación del fichero
Último acceso	Fecha del último acceso al fichero (S.O)
Propietario	Propietario del fichero asociado al evento
Tamaño	Tamaño del fichero asociado al evento.
Hash renombrado	Hash del path del fichero si para los eventos de renombrado
Ruta renombrado	Ruta absoluta del fichero si ha sido renombrado
Nombre renombrado	Nuevo nombre del fichero si ha sido renombrado
Secuencia temporal	Identificador único temporal asignado al fichero.

Tabla 3.2: Datos registrados

En el fichero CSV cada evento resultante es representado por una línea. Asimismo, cada uno de los datos de la línea se separan por medio de “;” Gracias a este formato, cuando un fichero CSV es leído por un software orientado a datos como Excel, RStudio o similar, el fichero es representado como una tabla donde las variables se ven por columnas y los eventos por filas.

```
51ae0330-4ec2…;2017-06-01 06:17:03+0000;Proyecto 1;LastWrite
04323270-4ec2…;2017-06-02 06:04:04+0000;Carta 1;LastAccess
c26fc910-4ec1…;2017-06-01 10:34:02+0000; IMG_20170119_201751;LastWrite
```

Figura 3.7: Ejemplo representación de evento en CSV

ID DEL EVENTO	FECHA DEL EVENTO	NOMBRE DEL FICHERO	WATCHER	…
51ae0330-4ec2…	2017-06-01 06:17:03	Proyecto 1	LastWrite	…
04323270-4ec2…	2017-06-02 06:04:04	Carta 1	LastAccess	…
51dcdb60-4ec2…	2017-06-01 10:34:02	IMG_20170119_201751	Size	…

Tabla 3.3: Ejemplo de representación de eventos en tabla

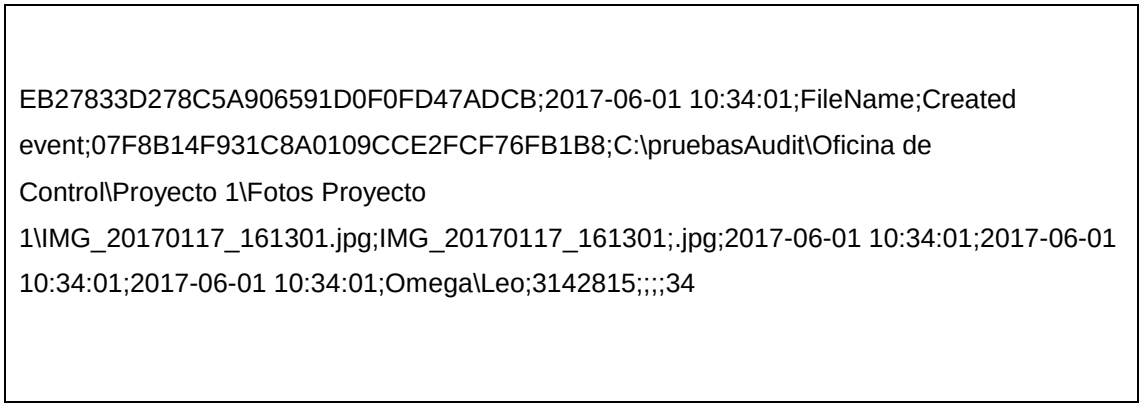

Figura 3.8: Extracto real del log de eventos

Eventos destacados

Como se ha visto en los apartados anteriores, hemos hecho referencia a varios tipos de eventos. Los eventos del sistema de ficheros, recogidos por FileSystemWatcher, los eventos de auditoria de ficheros de Windows, recogidos por EventLog y los eventos resultantes construidos por nuestra aplicación. Teniendo en cuenta que estos últimos resultan particularmente importantes para el presente trabajo, los designaremos simplemente como eventos.

Como hemos explicado anteriormente hay gran cantidad de combinaciones posibles entre los Filtros de notificaciones asociados a los Watchers y el tipo de acción asociado al propio filtro. En la documentación oficial de la librería FileSystemWatcher no encontramos información suficientemente detallada sobre qué acciones son llamadas para cada uno de los Filtros. Por ello ha sido tarea empírica comprobar dentro de las operaciones cotidianas cuales eran las acciones pertinentes para cada uno de los filtros. Pasamos a detallarlas en la siguiente tabla.

Filtro (NotifyFilter)	Acciones	Acción detectada
Attributes	Changed	Registra cambios en los atributos (Solo lectura, oculto y atributos avanzados)
CreationTime	Changed	Registra la creación de nuevos ficheros
LastAccess	Changed	Registra el acceso a un fichero
LastWrite	Changed	Registra una escritura en el fichero
Size	Created, Deleted, Renamed	Registra cambios en el tamaño del fichero
FileName	Created, Deleted, OnRenamed	Registra la creación de ficheros el cambio de nombre y el borrado
DirectoryName	Changed, Deleted, Renamed	Registra la creación de directorios, el cambio de nombre y el borrado

Tabla 3.4: Combinaciones seleccionadas para Watchers

En la tabla podemos apreciar que el filtro “FileName” por ejemplo es accionado para tres tipos de acciones diferentes Creates, Deletes y Renames. Esto se traduce en que cada vez que se crea un fichero nuevo Se produce la combinación (FileName, Created Event), cuando un fichero es borrado se produce la combinación (FileName, Deleted Event). Todas estas pruebas han sido realizadas con operaciones básicas del sistema.

Asimismo, debemos destacar que Windows denomina EVENTOS a la segunda columna, pero nosotros hemos optado por llamarla acciones, para evitar posibles confusiones.

Renombrado de ficheros

Debemos destacar que el renombrado de ficheros se presentó como una operación especial. Cuando se renombra un fichero, para el sistema uno de los ficheros desaparece mientras que otro nuevo se crea sin tener entre ellos vinculación alguna. Para poder seguir el historial de un fichero concreto hemos tenido que añadir campos especiales para este tipo de eventos. Estos campos son la ruta del fichero nuevo, el nombre y el hash creado. De esta forma podemos seguir buscando al mismo fichero a través de estos datos a partir de este punto en el tiempo.

Borrado de ficheros

El borrado de ficheros también ha presentado un reto, puesto que no se pude pedir datos de un fichero que ya no existe. Además, no se puede predecir en que momento sucederá un borrado ni tampoco podemos interceptarlos con las tecnologías elegidas. Hemos aprovechado todos los datos que se pueden generar a través de la ruta del fichero y del tiempo en el que se genera el evento, sin embargo, no se han podido guardar datos como la última modificación, último acceso u otros metadatos. Sin embargo, no creemos muy importante contar con estos datos pues la forma de uso esperada para nuestro programa es para un período de tiempo prolongado y por tanto tendríamos toda esta información ya guardada en la base de datos.

Eventos inesperados

En la fase de prueba de nuestro software hemos podido comprobar que para cada acción simple del usuario se sucede una serie de eventos, en los cuales se descompone que dicha acción.

Conjuntamente con los eventos esperados, se suceden otros asociados al funcionamiento de los diferentes programas. Generalmente esto ocurre con software de terceros que pueden hacer un uso de nuestros ficheros diferente al que cabría esperar. A continuación, detallaremos algunos de los lugares en los que han sido detectados estos comportamientos.

Gestión de ficheros en carpetas de Windows

En función del estado de la memoria de Windows y de otros mecanismos internos cuando accedemos a una carpeta por primera vez en una sesión/período de tiempo, Windows puede necesitar realizar una serie de acciones. Ejemplo de ello puede ser actualizar las miniaturas de los iconos, los metadatos de acceso o precargar cierta información. Todas ellas se producen de forma automática y no siempre bajo las mismas circunstancias.

Microsoft Office

Es conocido que Microsoft Office utiliza ficheros temporales para los documentos en los que está trabajando. Sin embargo, en nuestras pruebas hemos podido comprobar que no se utiliza un solo fichero por documento sino muchos diferentes. Por norma general estos ficheros van cambiando con el uso y no siguen un patrón de nombres determinado.

Software de Compresión

Otro de los comportamientos que es interesante destacar es el de ciertos softwares de compresión como el WinRar con el que hemos realizado pruebas. Puesto que WinRar utiliza carpetas temporales en otros directorios parte de la actividad no era guardad ya que estaba fuera de las rutas de monitorización programadas.

Por todo ello creemos que es crucial ir adquiriendo conocimiento empírico, a través de la toma y la observación de los datos obtenidos. De esta manera podremos introducir algoritmos de limpieza de los datos adecuados para minimizar el ruido generado por software de terceros y el propio sistema operativo.

3.4. Servidor

3.4.1. Diseño del funcionamiento

En esta segunda parte expondremos las diferentes partes de las que está compuesta nuestro servidor. Puesto que necesitamos realizar tres funciones bien diferenciadas contaremos con una parte por cada función. Las partes en las que hemos dividido el servidor son: Sistema de almacenamiento, sistema de análisis y sistema de representación.

Ahora bien, sabiendo los subsistemas de los cuales nos valdremos, pasaremos a explicar la relación existente entre ellos. Una vez nuestra base de datos (subsistema de almacenamiento) cuente con datos suficientes para ser analizados, el subsistema de análisis realizará las consultas necesarias y las operaciones propias para generar una digestión, que será enviada al subsistema de representación (servidor web) para su exposición al usuario final.

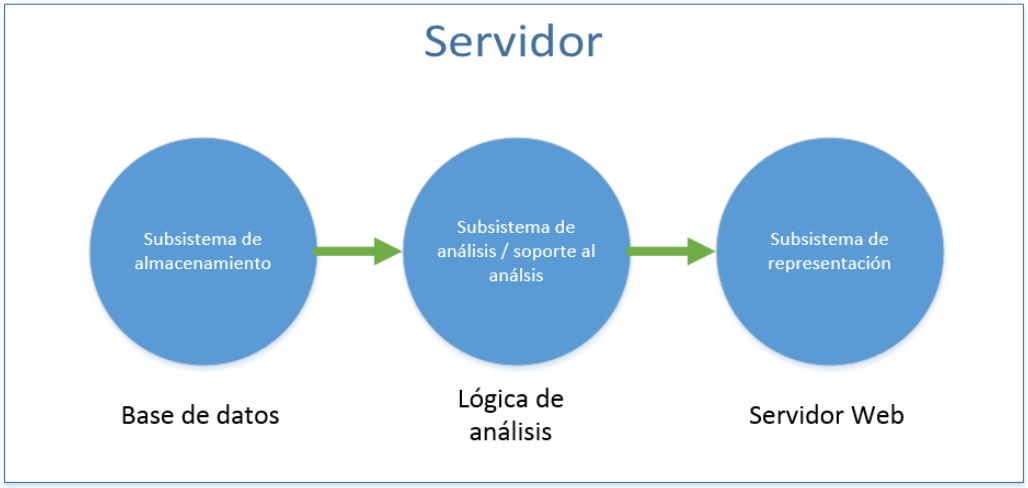

Figura 3.9: Esquema del servidor

3.4.2. Subsistema de Almacenamiento

El sistema de almacenamiento se encargará de almacenar los datos, organizarlos y permitir la consulta de los mismos a posteriori. Nuestro sistema de almacenamiento será una base de datos. La base de datos alimentará al subsistema de análisis.

3.4.2.1. Elección de la base de datos

Atendiendo al formato de los datos recogidos y a la presumible cantidad de los mismos una vez que el software sea utilizado por una base de usuarios importantes, rápidamente nos damos cuenta de dos cosas:

- El formato adoptado para los datos es totalmente lineal, en el sentido que no existen muchos tipos diferentes con relaciones complejas (Paradigma clásico de base de datos relacional). Tampoco son muy relevantes las relaciones más allá de qué evento le pertenece a qué usuario.

- La cantidad de datos que debería soportar un software de este tipo podría considerarse dentro de los límites del "Big Data" si la base de usuarios fuese suficientemente grande.

Tras una fase de recogida de información sobre los sistemas de bases de datos existentes y su afinidad para sortear problemas similares al que se nos plantea, hemos decidido que utilizaríamos una base de datos de la familia "NO SQL".

Este tipo de bases de datos surgen de la necesidad de grandes compañías de manejar ingentes cantidades de datos, que en su mayoría son simples. Características esenciales de ellas son la velocidad, la escalabilidad, la tolerancia a fallos y la integración con herramientas de análisis y/o representación.

Además, tenemos que tener en cuenta la estructura que tienen nuestros datos. Si hubiese que ordenarlos de alguna manera para almacenarlos la forma más lógica, esta sería cronológicamente puesto que representan una historia.

Teniendo en cuenta que necesitábamos una base de datos con capacidad de gestionar grandes cantidades de datos, que no era crucial la estructura relacional y que debía adecuarse o ayudar a gestionar series temporales, hemos decidido que la base de datos más adecuada es Apache Cassandra. A partir de ahora nos referiremos a ella simplemente como Cassandra.

Es necesario precisar que somos conscientes de que para el desarrollo de un prototipo no es estrictamente necesario utilizar una base de datos de estas características. Sin embargo, tras la fase de investigación comprendimos que, en algunos apartados, la

diferencia de metodología y modelado es muy grande. Para evitar desarrollar un prototipo sobre el que más tarde se tengan que hacer modificaciones de concepto, hemos decidido utilizar directamente la misma base de datos que utilizaríamos si el sistema pasase a producción.

3.4.2.2. Características específicas de Cassandra

Como ya hemos esbozado en el apartado anterior Cassandra posee una serie de cualidades que la hacen especialmente buena para el proyecto. Pasamos a detallarlas más abajo.

Afinidad a series temporales:

Cassandra es en circunstancias normales la base de datos elegida para gestionar series temporales. La estructura física sobre la que guarda los datos de forma consecutiva en disco permite búsquedas extremadamente rápidas aprovechando la localidad espacial y la capacidad de los discos duros mecánicos clásicos de leer masivamente datos contiguos.

Estructura tabular no regular:

Si atendemos al modelo de tablas que propone Cassandra, no será necesario tener una estructura fija. Es decir, Cassandra permite añadir filas a una tabla con mayor o menor número de columnas independientemente de la definición de la tabla. Consideramos que esta propiedad especialmente valiosa cuando tratamos con datos del sistema operativo, que generalmente son irregulares.

A pesar de ello, hemos seguido buenas prácticas de programación, definido la estructura de datos a utilizar y nos hemos mantenido con ellos.

Diseño especial para BigData:

El diseño específico para gestionar grandes volúmenes de datos está asociado a una serie de características que hacen de Cassandra una elección interesante, detallamos las más importantes.

- **Descentralización:**
  La base de datos desde el principio está planteada con un modelo multi-nodo. Todos los nodos tienen el mismo rol por lo que no hay un punto único de fallo. Así mismo soporta agrupaciones de nodo por localización permitiendo mapear

servidores físicos a nodos virtuales por racks, clusters y centros de datos. De esta forma la base de datos tiene conocimiento del diseño físico.

- **Replicación:**
  Cassandra soporta replicación dentro de clusters y si fuese necesario entre centros de datos. Es imprescindible destacar que la posibilidad de replicación aparece directamente como una práctica necesaria de diseño y no como un añadido a posteriori como en otros sistemas de bases de datos.

- **Tolerancia a fallos:**
  Puesto que es una base de datos descentralizada y que soporta replicación, la tolerancia a fallos está prácticamente asegurada. Cuenta con un modelo de comunicación entre nodos inteligente por el cual si se produce un fallo en un nodo se busca al siguiente que contiene la información buscada. Una vez que el nodo con fallos haya sido reparado la inclusión del mismo es automática a través del protocolo de comunicación, normalmente denominando “cotilleo” (del inglés gossiping), que hace que sus pares “actualicen” su información para hacerla coherente con el resto de nodos.

- **Adecuación para sistemas de análisis y representación:**
  Cassandra cuenta con drivers de conexión para los principales lenguajes de programación y módulos de conexión con sistemas de análisis como pueden ser Spark o similares. Nosotros hemos utilizado el driver de conexión a través del lenguaje Python pero creemos que la posibilidad de extender el sistema de análisis con otros paquetes de software especializados es un gran elemento a favor.

3.4.2.3. Diseño lógico de nuestra base de datos:

Se ha decidido configurar la base de datos con 3 nodos diferentes. De esta manera podemos aprovechar las características antes comentadas de replicación, tolerancia a fallos y la descentralización. [37, 38]

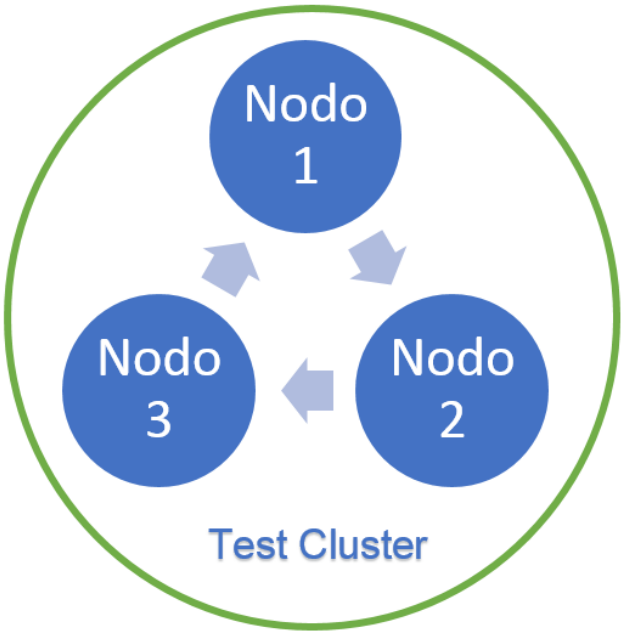

Figura 3.10: Test Cluster

Para cada uno de los nodos mencionados se ha generado una máquina virtual y un contenedor. Dentro del cual se ejecutarán nuestros nodos. [39]

En la fase de creación del contenedor se han utilizado los comandos especiales suministrados por el proveedor de la imagen de sistema utilizada, con ellos se establecieron las relaciones entre los nodos y otro tipo de configuraciones.

Docker fases relevantes:

Por el formato de virtualización de Docker, será necesario que nos descarguemos la imagen de sistema preparado para Cassandra. A partir de esta imagen se crearán los distintos contenedores.

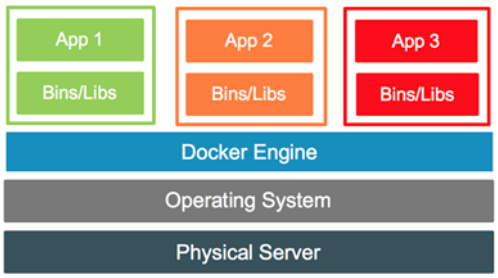

Figura 3.11: Esquema estructura de contenedores

Como hemos mencionado antes descargamos en cada máquina la imagen de Casandra de los repositorios de Docker “Docker Hub”. Utilizaremos la versión 3.10.

```
docker pull cassandra:3.10
```

Figura 3.12: Docker, descarga de la imagen de Cassandra

Creamos tres directorios en donde se guardará la información de cada nodo para que sea persistente. Siendo N el número de máquina virtual/contenedor asociado.

```
mkdir /home/leo/cassN_data
```

Figura 3.13: Directorios de persistencia de dato

Buscamos dentro del fichero "DockerFile" los puertos que tenemos que exponer del contenedor. En este caso el 7000, 7001, 7199, 9042 y 9160. Así como el sitio donde se guardará la información en el contendor para sobrescribir el punto de montaje /var/lib/cassandra

Creamos el contenedor:

```
docker create --name=cass1 -e
CASSANDRA_BROADCAST_ADDRESS=192.168.3.230 -v
/home/leo/cass1_data:/var/lib/cassandra  -p 7000:7000 -p
7001:7001 -p 7199:7199 -p 9042:9042 -p 9160:9160 --net=bridge
cassandra:3.10
```

Figura 3.14: Creación primer nodo

Detalle	
cass1	el nombre del primer nodo
192.168.3.230	la dirección ip del primer nodo
--net=bridge	red en modo bridge
-p 7000:7000 -p 7001:7001 -p 7199:7199 -p 9042:9042 -p 9160:9160	exponemos los puertos con -p
cassandra:3.10	utilizamos la versión de Cassandra 3.10
data:/var/lib/cassandra	ponemos el punto de montaje para que los datos sean persistentes fuera del contenedor

Tabla 3.5: Detalle de parámetros de creación de contenedor

Ahora que ya tenemos creado el primer nodo, crearemos los otros dos de forma similar para que se puedan conectar entre sí.

```
docker create --name=cass2 -e
CASSANDRA_BROADCAST_ADDRESS=192.168.3.229 -e
CASSANDRA_SEEDS=192.168.3.230 -v
/home/leo/cass2_data:/var/lib/cassandra -p 7000:7000 -p 7001:7001 -p
7199:7199 -p 9042:9042 -p 9160:9160 --net=bridge  cassandra:3.10
```

Figura 3.15: Creación segundo nodo

El segundo nodo con dirección ip 192.168.3.229 y escuchando al primero a través de la variable de entorno CASSANDRA_SEEDS.

Lo mismo para el tercer nodo. En el caso del tercer nodo solamente le tendremos que poner la dirección del primero o del segundo puesto que después entre ellos harán intercambio de información para conocer los nodos restantes.

Creación del Keyspace:

En Casssandra el contendor de datos más amplio es denominado KeySpace. Podemos asemejar el concepto de keyspace al de una única base de datos relacional, ya que dentro de él se encontrarán las familias de columnas (nomenclatura propia de Cassandra) o lo que normalmente se denomina como “tabla” para el resto de bases de datos.

El keySpace proporciona no solo un espacio común para las tablas de una aplicación (práctica recomendada), sino que permite la configuración de algunos parámetros especiales como son el factor de replicación, que afectará a todas las tablas que contenga.

```
CREATE KEYSPACE memento_ks
WITH replication = {
      'class' : 'SimpleStrategy',
      'replication_factor' : 2
};
```

Figura 3.16: Creación del Keyspace

Aquí se puede observar que hemos utilizado un factor de replicación “2”. Contando con tres nodos, esto significa que cada nodo contendrá aproximadamente el 66% de la información, permitiendo de este modo, seguir operando si alguno de los nodos fallase.

Los datos que escribimos en Cassandra son partidos automáticamente y dirigidos al nodo correspondiente elegido por el particionador “partitioner”, través de un token generado sobre la clave primaria. [40]

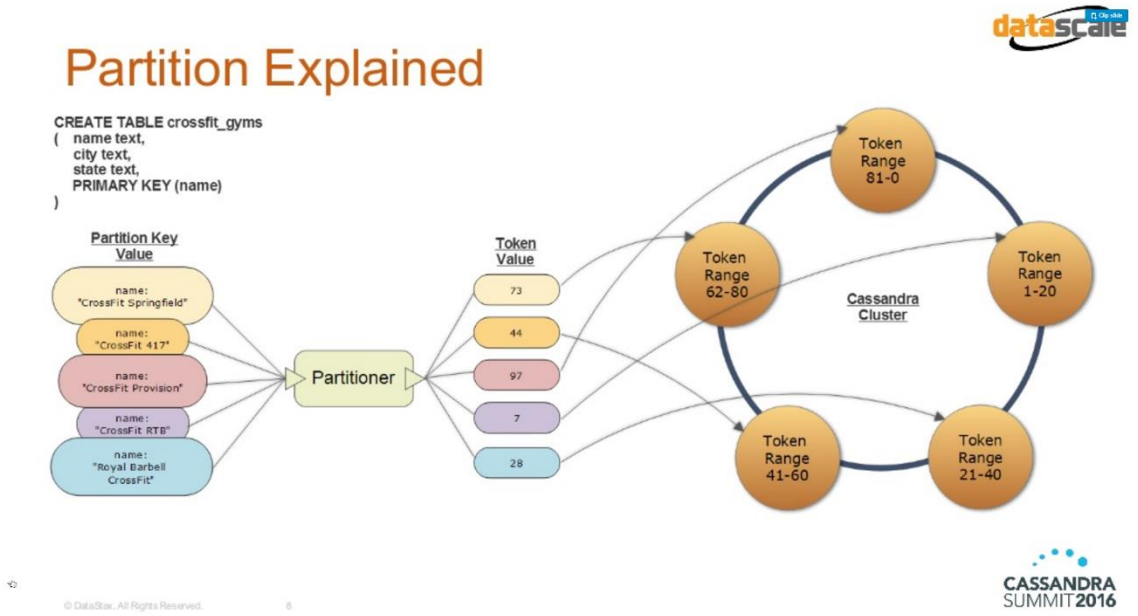

Figura 3.17: **(Grafica extraída de Cassandra Summit 2016, presentación de Adam Hutson)**

Podemos ver el particionado de nuestros datos en función de la replicación utilizada con la herramienta “nodetool”

```
leo@ubuntuServer1:~$ docker exec cass1 nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID
Rack
UN  192.168.3.228  961.86 KiB  256          65.8%
cbd6ea5b-3a87-458c-9ff7-cf4056394659  rack1
```

```
UN  192.168.3.229  1.04 MiB   256          71.3%
d9817769-5300-4c93-848e-3d33f4d4ea47  rack1

UN  192.168.3.230  1.08 MiB   256          62.9%
8af57687-eab6-4900-a93c-29456e919824  rack1
```

Figura 3.18: Nodetool

Creación de la tabla:

En Cassandra 3.1 las familias de columnas pasan a llamarse tablas como en el resto de bases de datos. Una vez creado nuestro KeySpace procederemos a crear una tabla que contendrá los datos de nuestra aplicación.

Como en cualquier otra clase de base de datos es importante familiarizarse con los tipos de datos que podemos utilizar. De especial importancia son los tipos de datos UUID y TIME_UUID. Estos funcionan como identificadores únicos para la clave primaria. Siendo Cassandra aun base de datos distribuida el hecho de poder generar claves primarias únicas simultáneamente no es un algo trivial. Por ello TIME_UUID es un tipo de dato especialmente importante, ya que nos permite crear claves en función del tiempo con una resolución muy elevada de tal forma que estemos completamente seguros de que no se puede crear otra clave idéntica en algún otro nodo simultáneamente.

Teniendo en cuenta todas las consideraciones de diseño de las fases anteriores, para nuestro trabajo hemos utilizado una tabla única a la que hemos llamado “principal”.

La tabla “principal” contiene un modelado de los eventos que son recogidos por el cliente.

PRINCIPAL		
Event_id	Time_uuid	K
User_hash	Text	K
Event_date	TimeStamp	C
Creation_date	timeStamp	
Event_type	Text	
Extensión	Text	
File_hash	Text	
File_size	Bigint	
File_name	Text	
Last_access_date	timeStamp	
Last_write_date	timeStamp	
Owner	Text	
Path	Text	
Ranamed_hash	Text	
Renamed_name	Text	
Renamed_path	Text	
Watcher	Text	
Seq_number	Int	

Tabla 3.6: Esquema chebokto de la tabla "Principal"

Código de creación de tabla principal, utilizado en el trabajo.

```
CREATE TABLE memento_ks.principal(
      event_id timeuuid,
      user_hash text,
      event_date timestamp,
      creation_date timestamp,
      event_type text,
      extension text,
      file_hash text,
      file_name text,
      file_path text,
      last_access_date timestamp,
      last_write_date timestamp,
      owner text,
      renamed_hash text,
      renamed_name text,
      renamed_path text,
      seq_number int,
      size bigint,
      watcher text,
      PRIMARY KEY (( event_id, user_hash ), event_date)
) WITH CLUSTERING ORDER BY ( event_date DESC )
```

Figura 3.19 Creación de la tabla principal

Dentro de cada partición de datos, éstos son ordenados en función de una clave secundaria denominada “Clustering key”, esto permite que los datos se ordenen en el momento de la escritura de tal forma que cuando se haga una consulta todos los datos aparezcan ordenados.

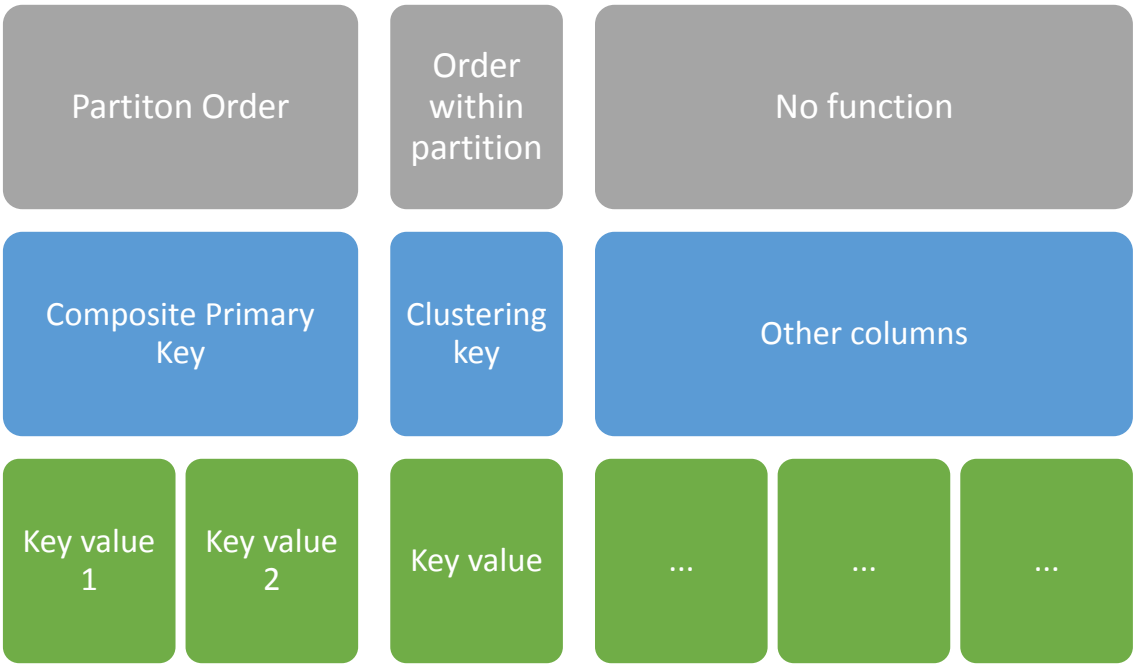

Figura 3.20: Esquema de ordenado modelo de los datos en Cassandra

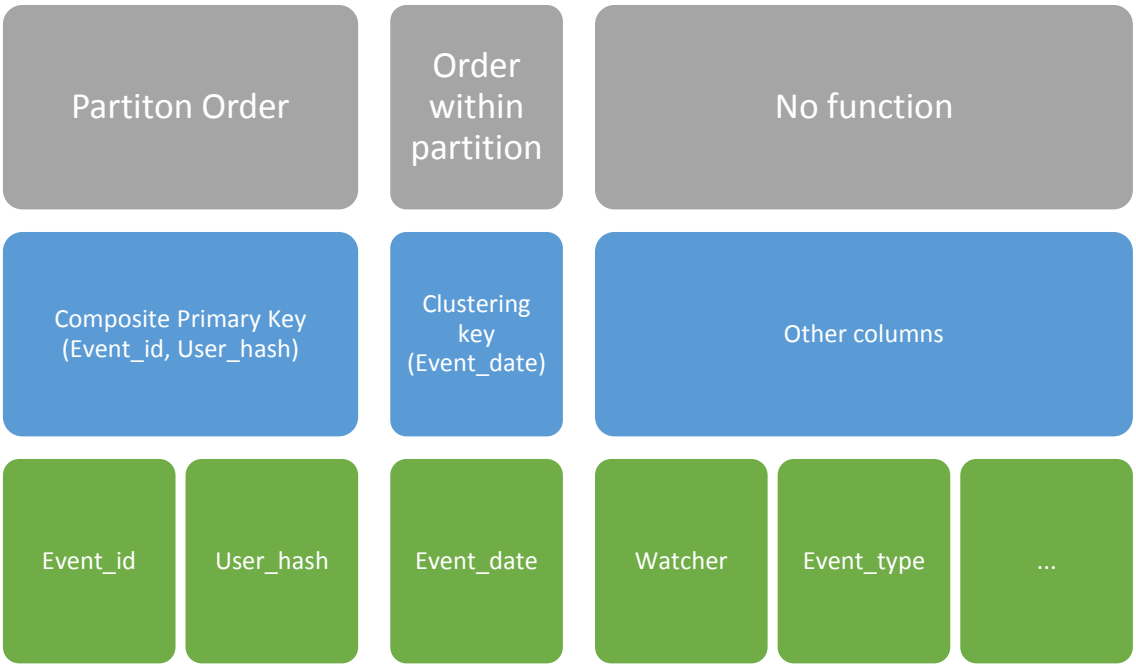

Figura 3.21: Esquema del orden elegido para nuestros datos

3.4.2.4. Volcado de los datos

El volcado de los datos a la tabla se realiza por medio de un Script Python [41]. Como en la fase de diseño se optó por utilizar Python para el análisis, lo más adecuado resulta hacer el volcado con el mismo lenguaje.

```
listContactPoints = ['192.168.3.230', '192.168.3.229', '192.168.3.228']
    bdConn = dbconnect.BDConn(listContactPoints)
    bdConn.connect()
    bdConn.create_schema()

    f = open(sys.argv[1], 'rt')
    try:
        reader = csv.reader(f)
        for row in reader:
            rowParts = row[0].split(';')
            print len(rowParts)
            print rowParts
            if(rowParts[12] == ''):
                rowParts[12] = 0
            if(rowParts[16] == ''):
                rowParts[16] = 0
            clq_query = """INSERT INTO memento_ks.principal (event_id,
user_hash, event_date, watcher, event_type,file_hash, file_path, file_name,
extension, creation_date, last_access_date, last_write_date, owner, size,
renamed_hash, renamed_path, renamed_name, seq_number) values (now(),
\'{}\', \'{}\', \'{}\', \'{}\',\'{}\', \'{}\', \'{}\', \'{}\', \'{}\',
\'{}\', \'{}\', \'{}\', {}, \'{}\', \'{}\', \'{}\',
{})""".format(*rowParts)
            print clq_query
            bdConn.insert_data(clq_query)
    finally:
        f.close()

    bdConn.close()
```

Figura 3.22: Script de volcado de datos

3.4.3. Subsistema de Análisis

El sistema de análisis contendrá la lógica necesaria para obtener resultados interesantes sobre los datos almacenados. En él se utilizarán métodos estadísticos, algoritmos clásicos y/o heurísticos según sea necesario. El sistema de análisis estará programado en Python.

3.4.3.1. Metodología

En una primera fase se procedió a la recogida de datos en un ordenador de control, con el objeto de obtener capturas tipo para su análisis.

Como ya hemos expuesto resulta necesario realizar un primer acercamiento a dichos datos, que nos proporcione un mejor entendimiento y que nos facilite la determinación de los métodos de análisis.

Para ello lo que hemos hecho es visualizar las capturas en crudo, como texto plano, después hemos cargado estos datos en RStudio (IDE para R), con el que hemos podido realizar transformaciones sencillas que pusieran de manifiesto patrones de interés.

Seguidamente se realizaron graficas de prueba para observar patrones, huellas y hacer comparaciones de resultados.

Para cada una de las pruebas realizadas se ha seguido de forma rigurosa el protocolo antes explicado.

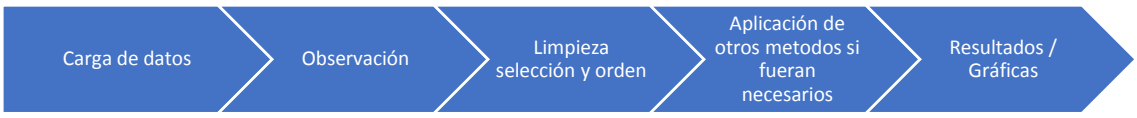

Figura 3.23: Esquema de pruebas

Contando con los resultados satisfactorios, se procedió a automatizarlos en Python para su posterior representación.

Puesto que hemos utilizado R para análisis inicial, hemos estado expuestos a conceptos/ herramientas como los dataframes, los vectores y las operaciones

vectoriales, que resultan especialmente adecuadas para este caso. Por ello al intentar automatizaren Python los procesos antes mencionados, nos hemos vistos privados de las herramientas referidas.

Como consecuencia de ellos hemos optado por continuar nuestra automatización de forma auxiliar la librería de Python Pandas.

Pandas sigue la misma filosofía de manejo de datos que R, encontrando nuevamente dataframes, vectores y operaciones vectoriales, lo que nos ha permitido implementar la automatización con las mismas herramientas teóricas que se hiciera en la fase de prueba, solo con cambios en la sintácticos.

3.4.3.2. Localización de las funciones de análisis:

El subsistema de análisis se encuentra físicamente dentro del subsistema de representación por conveniencia para el intercambio de los datos. Las funciones utilizadas se encuentran dentro del fichero utils_leo.py que a su vez hace uso de dbconnect.py que es una clase envoltorio para los drivers de conexión a la base de datos.

3.4.3.3. Subsistema de representación

El sistema de representación es parte fundamental del proyecto pues permite trasladar los resultados obtenidos al usuario final. En este apartado se pretende ser muy gráfico para que la información importante sea captada por el usuario con eficacia. Se utilizará un servidor web.

3.4.3.4. Elección del servidor web:

Para la elección del servidor web se ha tenido en cuenta principalmente dos cosas. La primera que fuese compatible con Python, pues el lenguaje utilizado en la fase anterior. La segunda que permitiese la creación de contenido dinámico con cierta facilidad.

Teniendo estos dos requisitos Django se presenta como el candidato óptimo para nuestro trabajo.

3.4.3.5. Funcionamiento del servidor web:

El funcionamiento de Django se basa en el modelo Vista-Controlador. Realiza un análisis sintáctico (parsing) de la url que el usuario le requiere y según sea ésta, ejecuta una o varias funciones de la vista.

Las funciones de la vista a su vez, pueden requerir la ejecución de otras funciones auxiliares normalmente relacionadas con la generación o consulta del contenido dinámico a mostrar.

Por último, se utilizan una serie de plantillas HTML combinadas con CSS y opcionalmente JavaScript (Templates), siguiendo la estructura general de una web actual. Dentro de estas plantillas encontramos etiquetas especiales destinadas a introducir el contenido dinámico. Estas etiquetas son reconocidas por el motor del servidor y son reemplazadas por el contenido real en tiempo de ejecución.

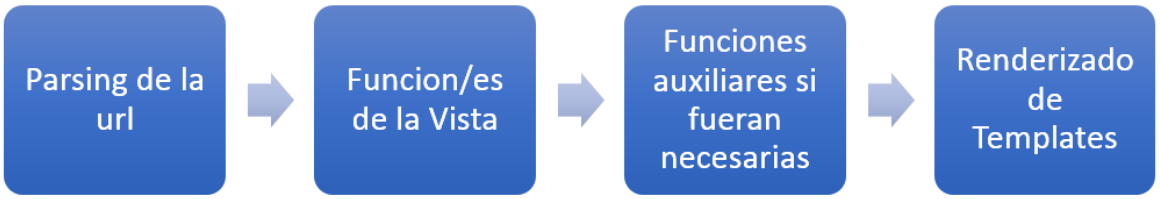

Figura 3.24: Esquena funcionamiento servidor web

A continuación, veremos extractos del código del servidor web para ilustrar el funcionamiento de este.

Ejemplo de fichero urls.py con las expresiones regulares con las cuales se analizarán las urls pedidas al servidor.

```
urlpatterns = [
    url(r'^$', views.Plot1DView.as_view(), name='plot1d'),
    url(r'^plot2$', views.Plot3DView.as_view(), name='plot3d'),
]
```

Figura 3.25: Reglas de parsing

Cuando al servidor se le pida la URL …/plot3d, urls.py llamará a la función as_view() de la clase Plot1DView alojada en la vista.

```
class Plot3DView(TemplateView):
    template_name = "memento_dashboard/plot3d.html"

    def get_context_data(self, **kwargs):
        # Call the base implementation first to get a context
        context = super(Plot3DView, self).get_context_data(**kwargs)
        variableAdevolver = utils_leo.activities3d()
        context['plot'] = {'uno': variableAdevolver }
        return context
```

Figura 3.26: Extracto de la vista

En la Figura 3.26 observamos, la llamada que hace la vista a nuestras funciones de análisis para que le devuelvan los resultados que serán inyectados en la página web.

```
{% extends "memento_dashboard/base.html" %}

{% block head %}
{{block.super}}
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
{% endblock %}

{% block content %}

<!-- /.container -->
<div>
        <a id="actividad"></a>
        <div class="w3-container" id="services" style="margin-top:75px">
            <h1 class="w3-xxxlarge w3-text-red"><b>Actividad</b></h1>
            <hr style="width:50px;border:5px solid red" class="w3-round">
        </div>
        <div class="fullscreen">
                {{plot.uno | safe}}
        </div>
</div>
<!-- /.container -->
{% endblock %}
```

Figura 3.27 Ejemplo de Template

En la Figura 3.27 se observan las etiquetas especiales que serán sustituidas por el contenido devuelto por la Vista. De ello se encarga el motor del servidor. Por último, el código HTML generado es enviado al usuario.

3.4.3.6. Acceso a los datos:

Para representar los datos almacenados en la base de datos, utilizamos las funciones auxiliares del fichero utils_leo.py llamadas por la Vista, quienes hacen consultas a la base de datos y los preparan. Luego estas funciones llaman a la librería de Plotly a quien le proporcionan los datos procesados y por último ésta le devuelve a la Vista una cadena de texto que realmente es un bloque HTML para insertar en la web.

3.4.3.7. Representación gráfica:

Para la representación de gráficas en la web hemos utilizado la librería Plotly. Esta librería que funciona en javascript, está especialmente indicada para realizar gráficas compatibles con tecnologías web. Además, nos provee de librerías para el lenguaje R y otras para Python y Pandas. El formato de salida de las gráficas generadas es una cadena de caracteres que tiene código HTML con estilos y javascript. Esta salida la podemos insertar directamente a través de las etiquetas incluidas en los templates de Django.

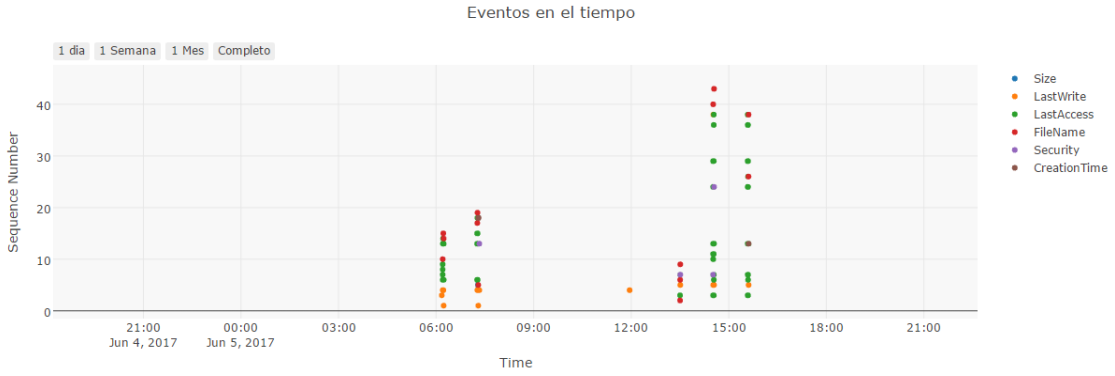

Figura 3.28: Ejemplo de grafica realizada con Plotly

3.4.4. Estructura del servidor

Ahora que ya hemos visto todas las partes que componen el servidor, creemos conveniente repasar la estructura utilizada y las relaciones ellas.

Para el servidor hemos utilizado cuatro máquinas virtuales. Tres de ellas para el sistema de almacenamiento y la cuarta para el sistema de análisis y representación.

Configuración física de las máquinas virtuales:

Máquina virtual 1	
Nombre Máquina virtual	UbuntuServer1
Función:	Cassandra Nodo 1
Sistema Operativo	Ubuntu Server XX
Núcleos	1
Memoria RAM	4GB
Almacenamiento	20GB
Dirección IP	192.168.3.230

Tabla 3.7: Configuración máquina virtual 1

Máquina virtual 2	
Nombre Máquina virtual	UbuntuServer2
Función:	Cassandra Nodo 2
Sistema Operativo	Ubuntu Server XX
Núcleos	1
Memoria RAM	4GB
Almacenamiento	20GB
Dirección IP	192.168.3.229

Tabla 3.8: Configuración máquina virtual 2

Máquina virtual 3	
Nombre Máquina virtual	UbuntuServer3
Función:	Cassandra Nodo 3
Sistema Operativo	Ubuntu Server XX
Núcleos	1
Memoria RAM	4GB
Almacenamiento	20GB
Dirección IP	192.168.3.228

Tabla 3.9: Configuración máquina virtual 3

Máquina virtual 4	
Nombre Máquina virtual	Ubuntu Desktop
Función:	Cassandra Nodo 1
Sistema Operativo	Ubuntu Desktop 16.04
Núcleos	2
Memoria RAM	4GB
Almacenamiento	20GB
Dirección IP	192.168.3.230

Tabla 3.10: Configuración máquina virtual 4

Configuración software de las máquinas virtuales:

Para las máquinas de la base de datos, se ha optado por crear contenedores con Docker como hemos visto anteriormente, mientas que para el servidor web hemos utilizado una instalación clásica.

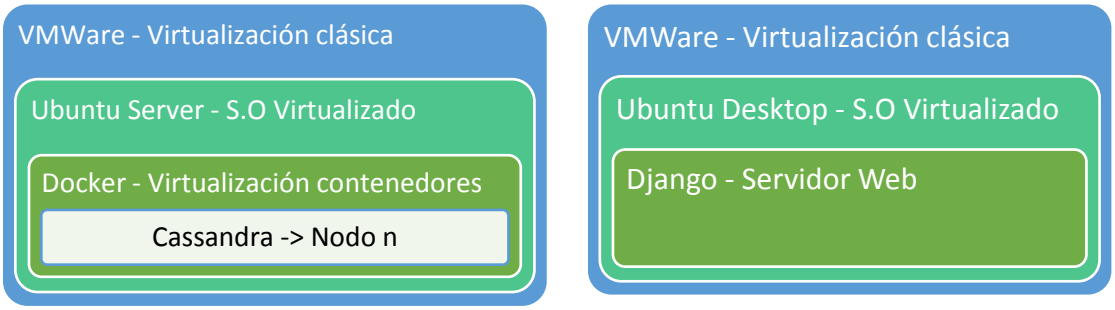

Figura 3.29: Esquema tipos de máquina virtual

En el siguiente gráfico vemos desde otra perspectiva la representación del software utilizado para cada una de las máquinas virtuales.

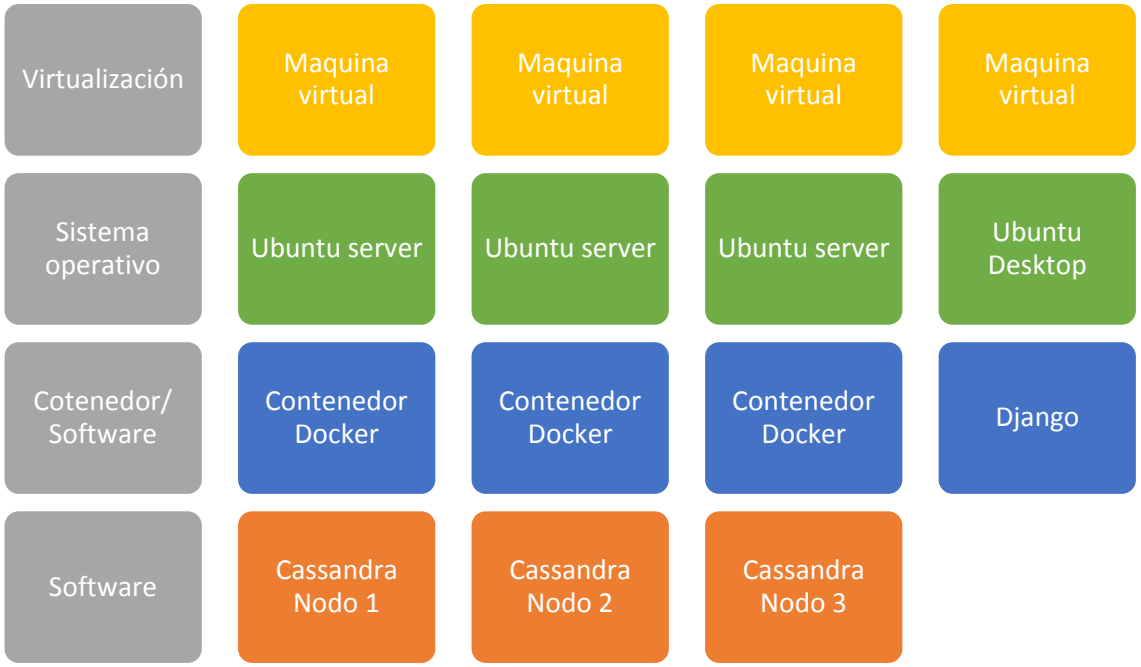

Figura 3.30: Esquema por niveles software de las 4 máquinas virtuales

3.4.5. Ejecución del servidor

Para las pruebas de todo el servidor se han utilizado dos ordenadores. El primero un ordenador de sobremesa en el que se ha hecho el desarrollo y el segundo un microservidor.

Configuración de las máquinas físicas:

Ordenador Escritorio	Microservidor
Procesador: i7-6700k 4.0Ghz	**Procesador:** Intel Celeron G1610T 2.3Ghz
Memoria RAM: 16GB	**Memoria RAM:** 10GB
Disco Duro: 1TB	**Disco Duro:** 1TB
S.O: Windows 8.1 Profesional	**S.O:** ESXi 5.5

Tabla 3.11: Hardware utilizado para ejecutar las máquinas virtuales

Distribución de las máquinas virtuales:

Debido al elevado consumo de memoria ram de los nodos de Cassandra (4GB/Nodo) hemos tenido que dividir las máquinas virtuales en los dos ordenadores arriba descritos.

- En el microservidor hemos ejecutado el nodo 1 y el nodo 2 de Cassandra.
- En el ordenador de escritorio hemos ejecutado el nodo 3 y la máquina con el servidor web.

Para la virtualización en el ordenador de sobremesa se ha utilizado VMWare Workstation mientras que la virtualización en el microservidor se ha hecho directamente sobre ESXi.

Las máquinas del microservidor se han creado en el escritorio y luego se han migrado a ESXi a través de las herramientas de vmWare.

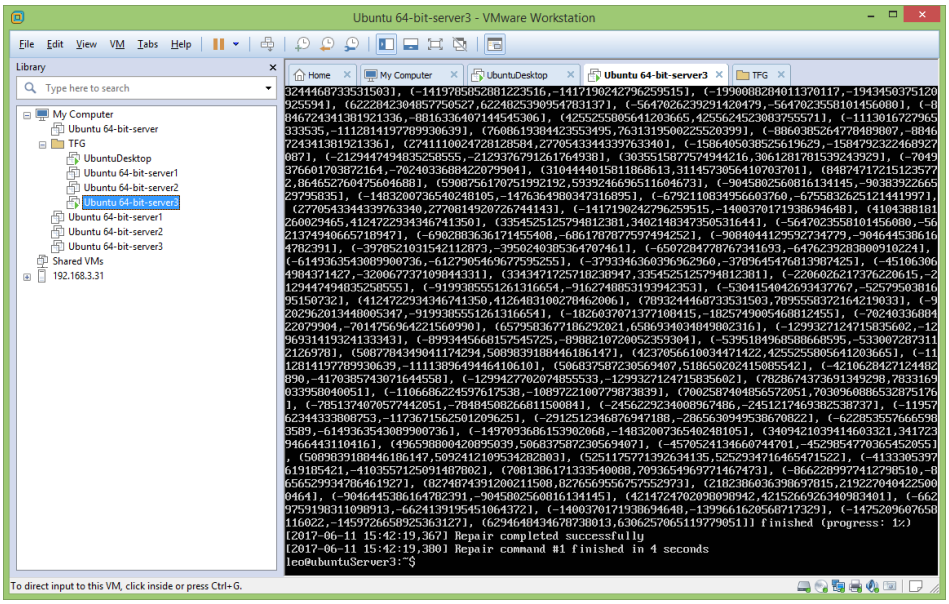

Figura 3.31: VmWare workstation en ordenador de escritorio

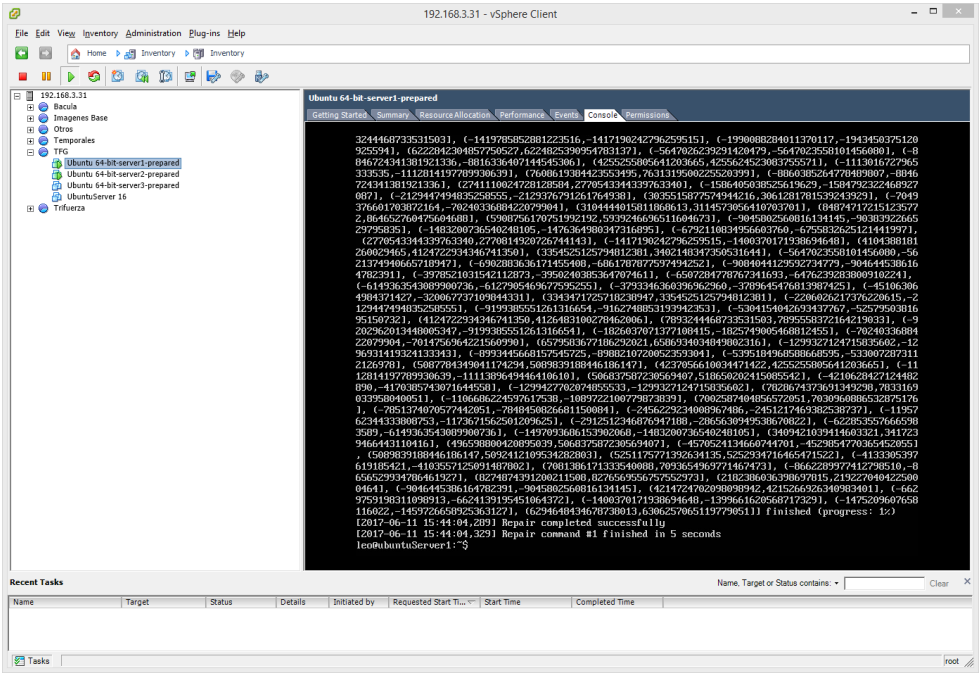

Figura 3.32: Cliente VSphere para el microservidor

4. Resultados de la monitorización

A continuación, expondremos los resultados de la monitorización y el análisis realizado mediante la infraestructura. Debido que las posibilidades de análisis de los datos son innumerables hemos escogido un grupo representativo para los objetivos del trabajo.

4.1. Información para los usuarios

Visualización de la actividad:

En las gráficas de actividad se ha representado el uso de los ficheros prestando especial atención a si estos son reutilizados a lo largo del tiempo o si por el contrario sólo se utilizan brevemente (localidad temporal).

Hemos utilizado el concepto de numero de secuencia temporal asociado a los ficheros en uso. A cada fichero cuando es utilizado por primera vez en la sesión, se le asigna un identificador único (número de secuencia). El objetivo de esta gráfica es poder visualizar rápidamente si está habiendo un uso continuado de los mismos ficheros o si por el contrario se están utilizando ficheros diferentes cada vez. A continuación, vemos un ejemplo:

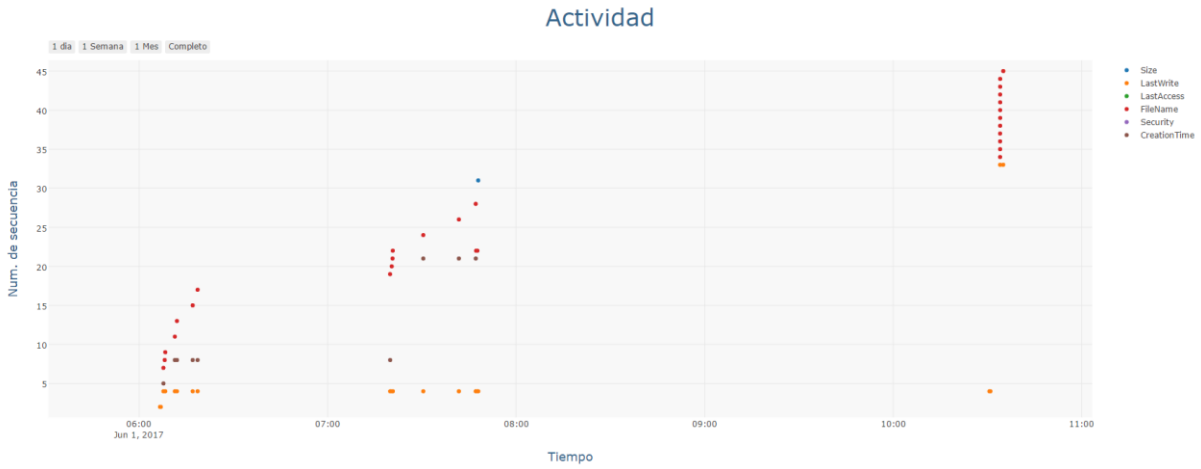

Figura 4.1: Primer ejemplo de grafica de actividad.

En la Figura 4.1 se puede observar como el usuario ha generado actividad a lo largo de 5 horas, de 6 de la mañana a 11 horas. En él se observan los diferentes tipos de operaciones generadas dentro del rango posible (Creación, Escritura, Lecturas…). La altura de la gráfica representa la cantidad de ficheros diferentes con los que ha trabajado el usuario en esa sesión. En este caso 45.

A continuación, pasaremos a analizar algunos ejemplos concretos de gráfica de actividad, para poder ilustrar la cantidad de inferencias que podemos obtener de ellas.

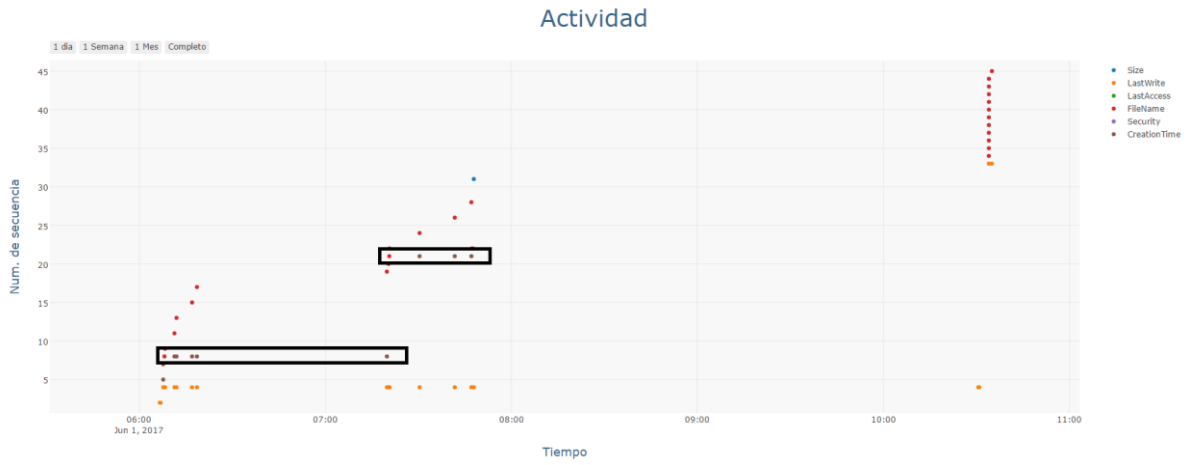

Figura 4.2: Primer ejemplo de gráfica de actividad (2)

En la Figura 4.2 podemos observar cómo se van produciendo una serie de acciones con el mismo número de secuencia. En este caso 8 y 21. Que se corresponden con la creación de dos nuevos ficheros de Excel y Word respectivamente. Ligeramente por encima de cada uno de ellos encontramos una diagonal de puntos rojos correspondiente a la creación de archivos temporales de office.

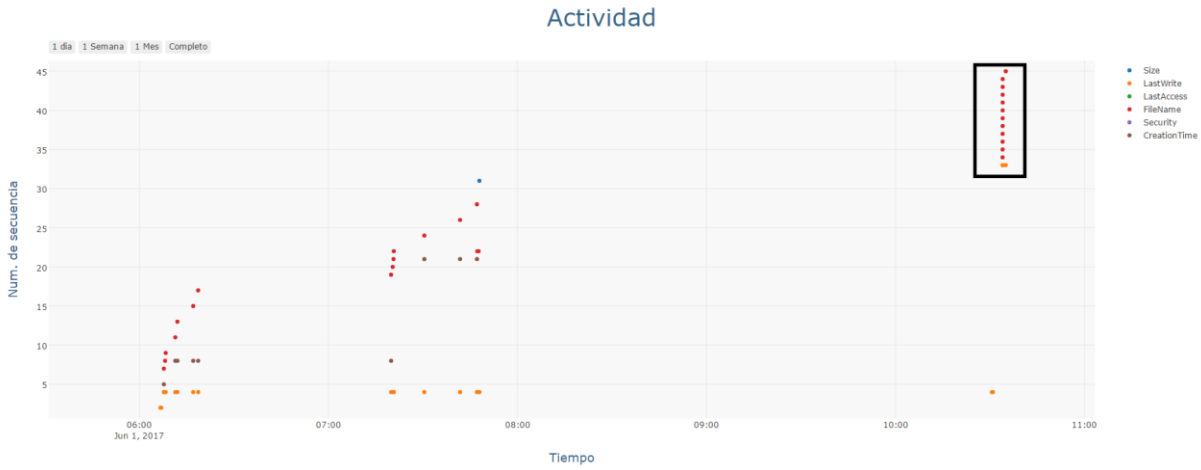

Figura 4.3: Primer ejemplo de gráfica de actividad (2)

Otro patrón interesante se aprecia en la Figura 4.3, que sucede entre las 10 y las 11 horas. Se producen una serie de eventos del tipo “FileName” y del subtipo “Created event”. Por lo tanto, estamos viendo la creación de muchos ficheros a la vez y a gran velocidad. Esta acción se corresponde con la copia de un grupo de fotografías desde una cámara a la carpeta monitorizada.

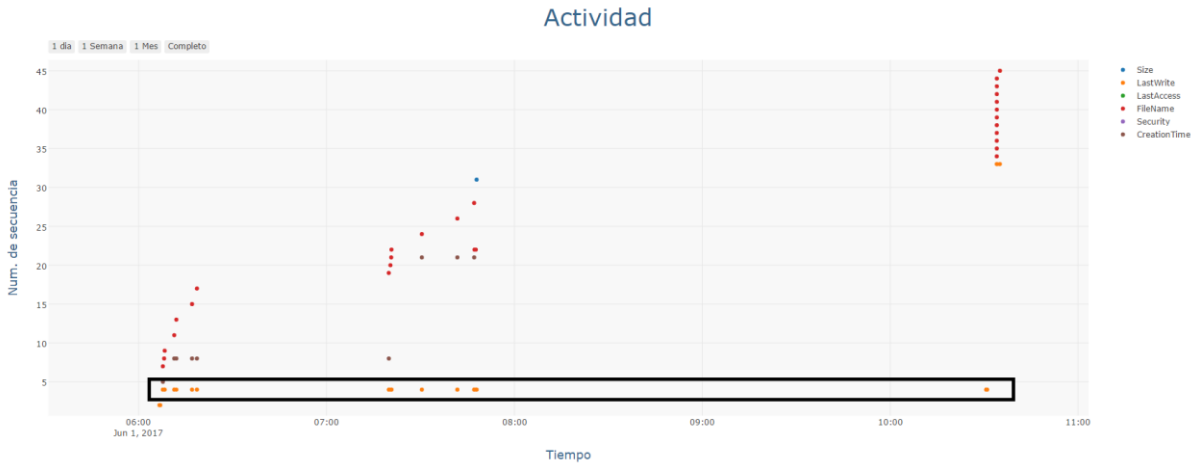

Figura 4.4: Primer ejemplo de gráfica de actividad (3)

Por último, en la Figura 4.4, podemos ver una serie de escrituras en los mismos puntos de tiempo en los que se producen las acciones que hemos explicado anteriormente, pero con un número de secuencia más bajo. Todas estas escrituras han sido registradas a la misma altura ya que pertenecen a la carpeta que contiene los ficheros que han ido cambiando.

Veamos otro ejemplo de interés:

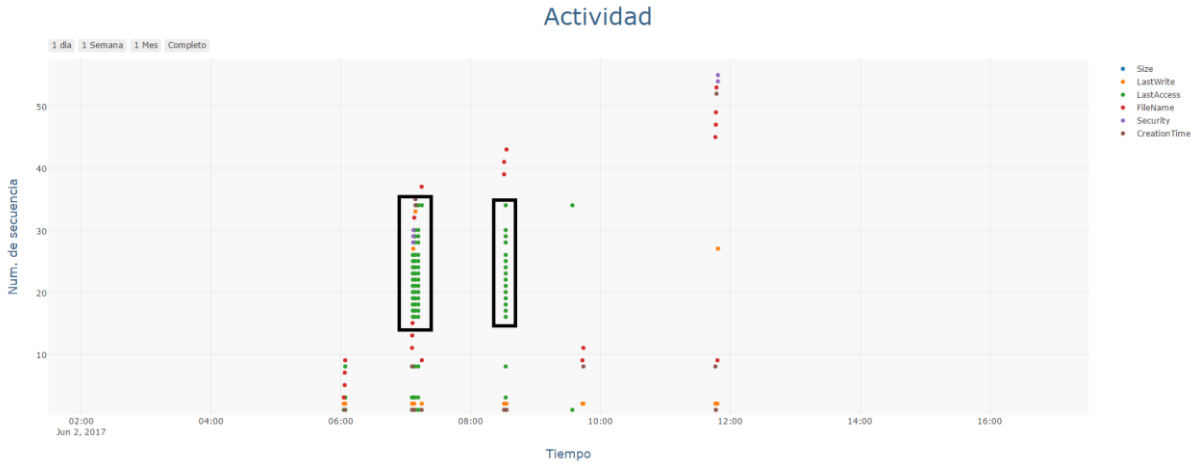

Figura 4.5: Segundo ejemplo de actividad.

En la Figura 4.5 podemos observar dos columnas de accesos suficientemente altas y estrechas como para pensar que el usuario no las ha generado con acciones singulares. Si observamos qué ficheros son, vemos que todos ellos son fotos de la carpeta de trabajo. Lo que sucede es que cada vez que uno entra a la carpeta donde se encuentran las fotos, éstas son accedidas por el sistema operativo para mostrar las miniaturas. A este tipo de patrones nos hemos referido anteriormente en el apartado de “Eventos inesperados” y en esta figura se aprecia con claridad.

Otro aspecto importante a destacar es que la resolución máxima utilizada para los “Time Stamps” de los eventos es de segundos, por ello algunos eventos aparecen solapados en las gráficas y solo son visibles cuando ocultamos algún Watcher. Por ejemplo, hemos observado que en la creación de un fichero se produce un evento “CreationTime” e inmediatamente después un evento “Security”. Otra pareja de eventos que se da con frecuencia es la de creación de un fichero temporal y acceso o escritura al mismo.

Visualización del incremento del espacio utilizado:

Para las gráficas de espacio utilizado se ha buscado tener una aproximación del crecimiento real que tienen las carpetas del usuario monitorizadas. Con ella podemos detectar crecimientos repentinos/inesperados del espacio necesario para copias de

seguridad. En conjunto con las gráficas de actividad podremos determinar que acciones o patrones se podrían optimizar en relación al espacio utilizado.

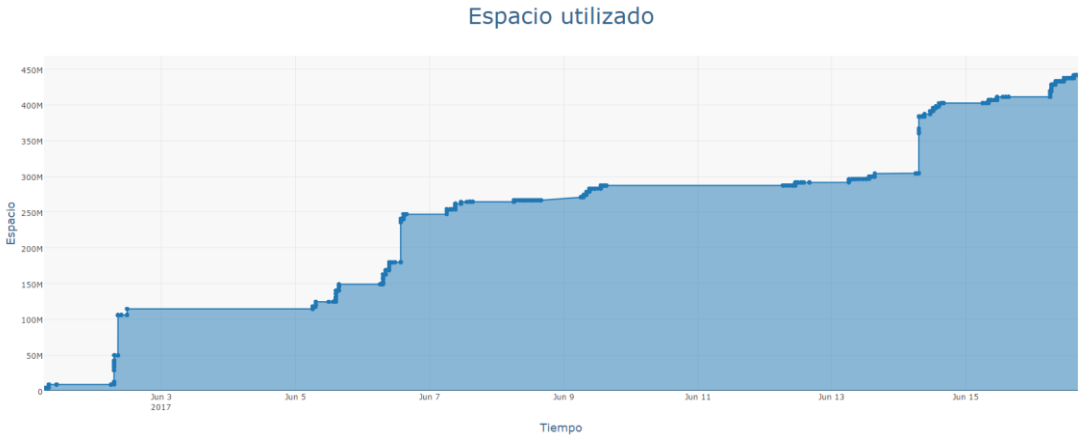

Figura 4.6: Gráfica espacio utilizado.

Visualización de los tipos de ficheros accedidos:

En la gráfica de tipo de ficheros se muestra el porcentaje de cada tipo de ficheros utilizados por el usuario. Con ella se puede ver rápidamente que tipos de ficheros trabajan normalmente los usuarios. En función de los ficheros utilizados podemos esperar un crecimiento proporcional al tipo de actividad. Como ejemplo podríamos decir que no es lo mismo trabajar habitualmente con documentos de texto que con fotografías en formato raw.

Figura 4.7: Gráfica Tipos de ficheros.

Visualización de escrituras/lecturas:

En las gráficas de escrituras/lecturas se muestra el acumulado de lecturas y escrituras para cada uno de los días del mes. Con ella podemos ver si existe balance o si hay una operación que se realiza más que la otra con el objetivo de hacer optimizaciones. Una posible optimización es la elección del tipo de almacenamiento a comprar, por ejemplo, si se realizan muchas escrituras presumiblemente el uso de un disco duro mecánico será lo más aconsejable puesto que un disco duro de estado sólido (SSD) tendría un desgaste muy elevado.

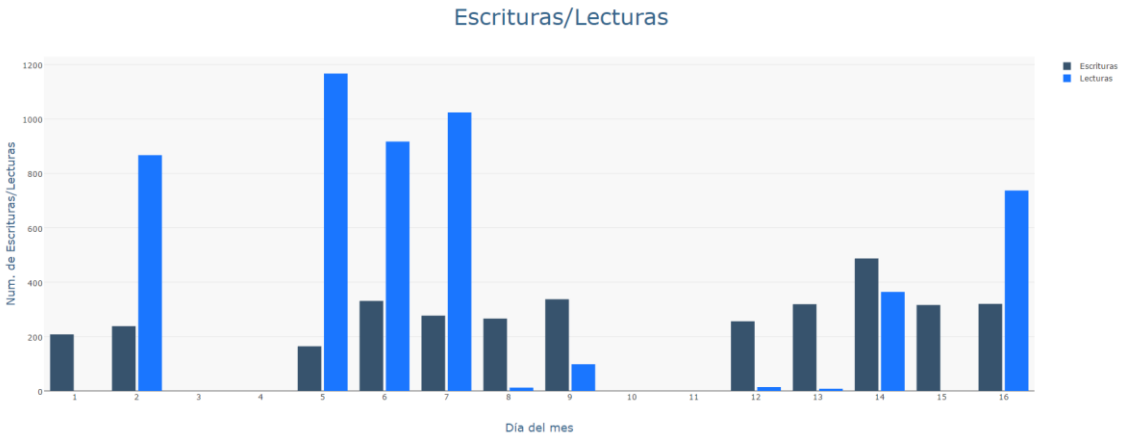

Figura 4.8: Gráfica lectura/escritura.

Visualización del índice de importancia:

El resultado más relevante de todo el trabajo es el índice de importancia. En él se puede ver una clasificación de importancia según el uso de los ficheros, que han sido monitorizados. Para llegar a este resultado se han aplicado técnicas de limpieza al conjunto de datos obtenidos. Con ellas se han filtrado eventos de ficheros temporales, ficheros ocultos y otras anomalías detectadas que no resultan de interés para esta tabla. Más tarde hemos calculado el índice de importancia de cada fichero en base a la cantidad de lecturas y escrituras relativas al total.

$$\text{Índice Importancia} = \frac{Accesos_{fichero}}{Accesos_{totales}} + \frac{Modificaciones_{fichero}}{Modificaciones_{totales}}$$

Figura 4.9: Fórmula del índice de importancia

Por último, los ficheros han sido ordenados de forma decreciente en función de este índice. De esta forma el usuario puede comprobar si los archivos que nosotros hemos clasificado como de mayor importancia también lo son para su actividad.

file_names	file_paths	extensiones
Presupuesto 1	C:\pruebasAudit\Oficina de Control\Proyecto 1\Pres	.xlsx
Presupuesto 2	C:\pruebasAudit\Oficina de Control\Proyecto 2\Pres	.xlsx
Carta 2	C:\pruebasAudit\Oficina de Control\Proyecto 2\Cart	.docx
Carta 1	C:\pruebasAudit\Oficina de Control\Proyecto 1\Cart	.docx
email_colaborador	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.rar
IMG_20170119_201751	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
IMG_20170117_161249_2CS	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
Carta 1	C:\pruebasAudit\Oficina de Control\Proyecto 1\Cart	.pdf
IMG_20170117_161301	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
IMG_20170112_202014	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
GROHE_TARIFA_EDICION	C:\pruebasAudit\Oficina de Control\Descargas\GROHE	.pdf
IMG_20170117_161249_2CS - copia	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
IMG_20170119_201751 - copia	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
IMG_20170112_190820	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg
IMG_20170119_201751 - copia (2)	C:\pruebasAudit\Oficina de Control\Proyecto 1\Foto	.jpg

Figura 4.10: Ejemplo de tabla de Importancia de ficheros.

Gráficas en el servidor web:

Para hacer el acceso y lectura de las gráficas más sencillo hemos optado por insertarlas en el servidor web. En la Figura 4.11 Más abajo mostramos los resultados a los que tendrían acceso los usuarios. Dado que este trabajo está orientado a PYMES y profesionales independientes hemos considerado esencial que las gráficas sean interactivas y que la plataforma en la que se fueran a mostrar fuese fácilmente accesible.

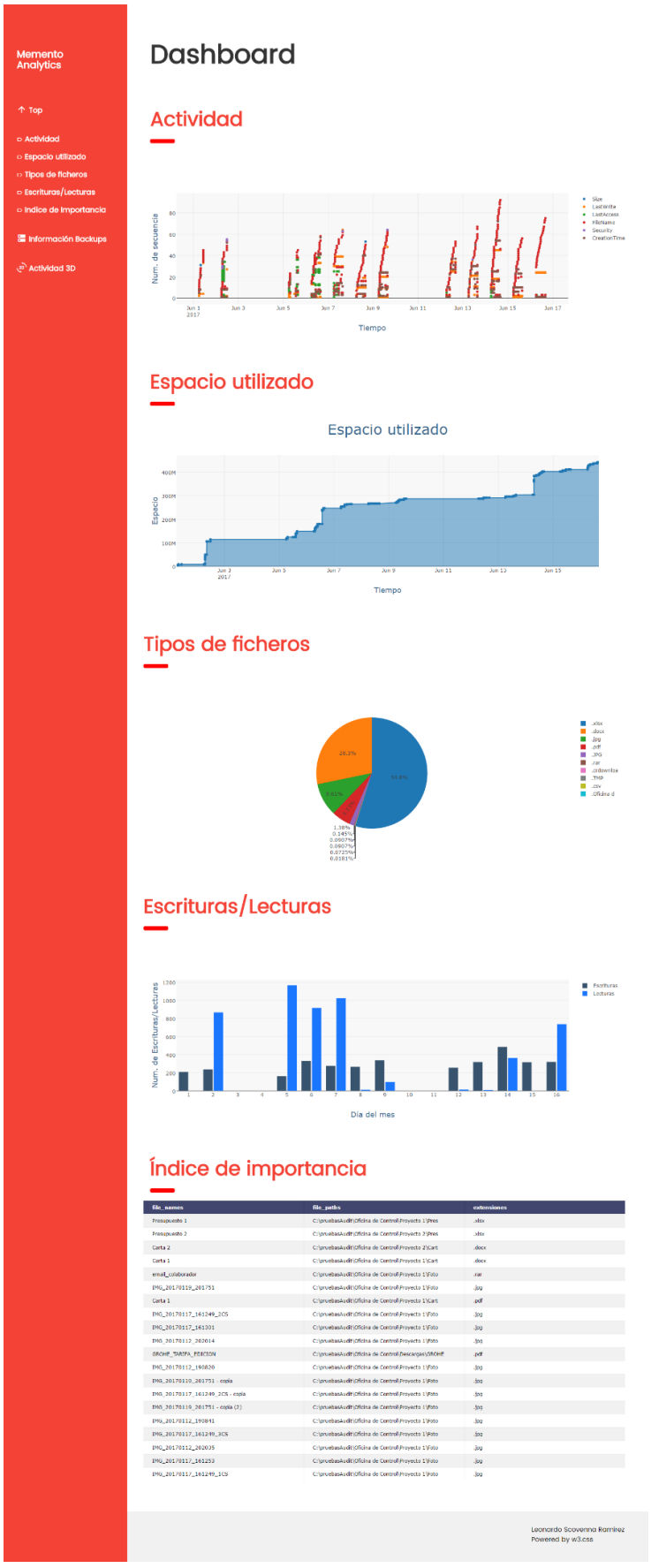

Figura 4.11: Gráfica web.

4.2. Aplicación a sistema de Backup

En base a los resultados obtenidos podemos ver que éstos son un conjunto de datos que fácilmente se traducen a conocimiento o reglas aplicables a un sistema de backup con el objetivo de optimizar los recursos.

Mediante la aplicación de este prototipo hemos podido identificar tres grandes grupos de ficheros. Los ficheros que pertenecen al “Working Set”, (Ficheros en uso) los ficheros temporales y los ficheros de Archivo. De esta manera y como hemos adelantado, podemos utilizar el modelo de backup por niveles en nuestro beneficio.

En el sistema de niveles de backup la copia de seguridad completa es representada por el nivel cero y las copias incrementales/diferenciales por niveles superiores. En cada nivel n no se incluye aquello que ya esté incluido en el nivel n-1. Por ello proponemos incluir a esta escala un nivel n = -1 de tal forma que en él se encuentren los ficheros que pertenezcan al archivo. Con este modelo evitaremos hacer copias de seguridad completas que contengan ficheros que no cambian o no son utilizados a lo largo de un período de tiempo determinado. De esta forma obtendríamos la reducción de espacio deseada, sin la necesidad de determinar manualmente que ficheros pertenecen a la clase archivo y cuáles no y además poder descartar los temporales.

En la página web hemos incluido un apartado denominado “Información para backups” en él podemos encontrar tres tablas que muestran la clasificación realizada para las categorías “Working Set”, archivo y temporales. Es posible descargar las listas de ficheros en texto plano para su aplicación directa a las rutas incluidas y excluidas para la copia de seguridad. Con esta información el usuario podrá optimizar sus recursos, teniendo una herramienta de ayuda a la toma de decisiones en sistemas de backup.

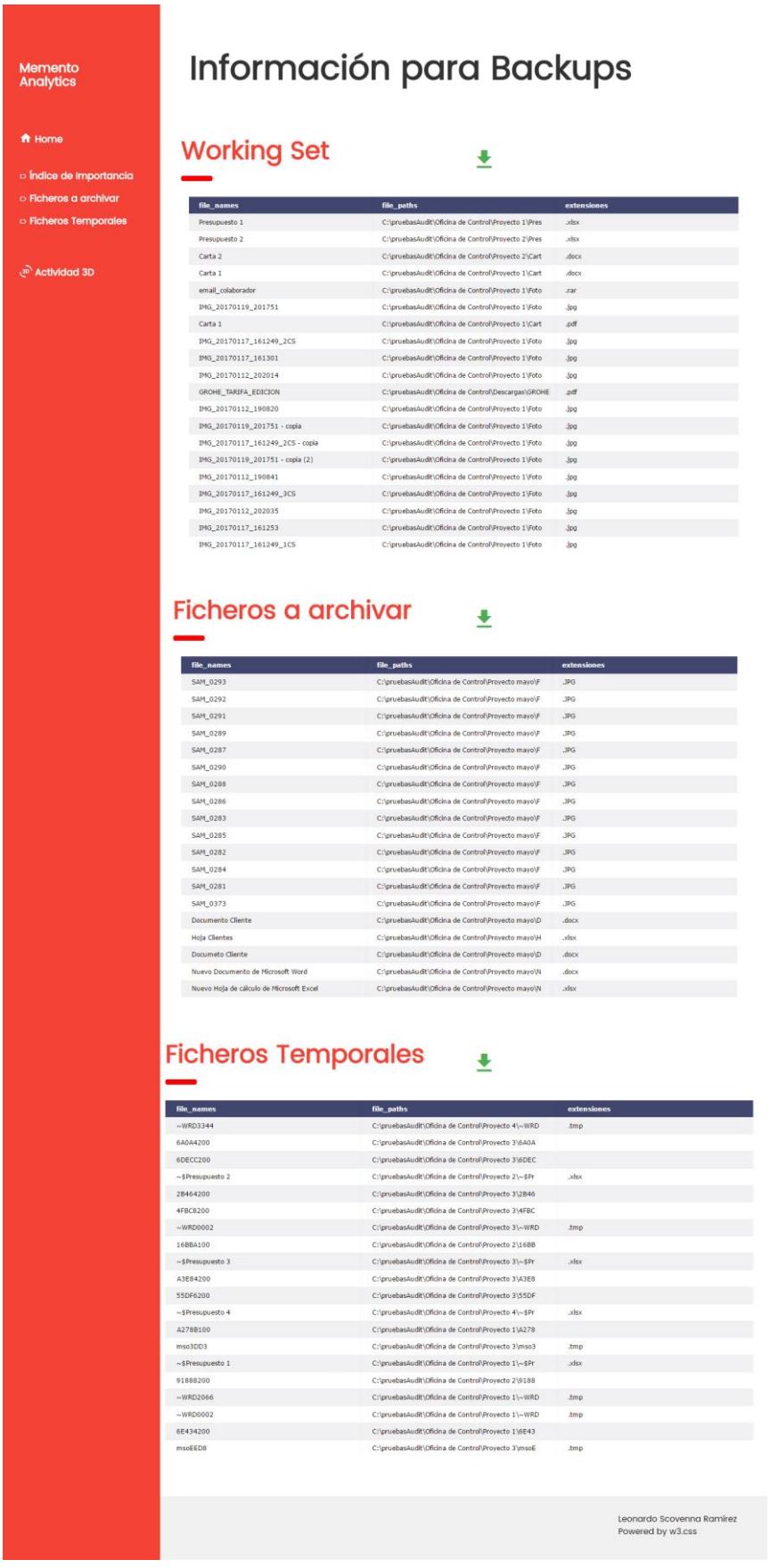

Figura 4.12: Ejemplo de tabla de Importancia de ficheros.

Veamos en la Figura 4.13 un supuesto de aplicación en donde a través de los comandos del software de backup incluimos la lista de ficheros del working set y excluimos los que pertenecen a la categoría de temporales. Así mismo generamos un backup completo de nivel inferior para los ficheros de archivo.

```
CopiaDeSeguridad --nivel=-1 --include=ficheros_archivo.txt
CopiaDeSeguridad --nivel=0 --include=ficheros_importantes.txt
CopiaDeSeguridad --nivel=3 --include=ficheros_importantes.txt --exclude=ficheros_temporales.txt
```

Figura 4.13: Posible aplicación sobre sistema de copia de seguridad.

5. Conclusiones

El presente trabajo se ha realizado con el objetivo de analizar la viabilidad de un prototipo de herramienta de monitorización y análisis que nos ayudara en la toma de decisiones para las copias de seguridad. En base a los resultados conseguidos hemos podido obtener las siguientes conclusiones.

Sobre los resultados obtenidos:

A priori los resultados obtenidos han sido satisfactorios. Hemos conseguido realizar un prototipo basado en las ideas y objetivos originales a pesar de las dificultades técnicas que han surgido a lo largo del desarrollo.

Consideramos que los resultados obtenidos son relevantes para la toma de decisiones relacionadas con las copias de seguridad para PYMES. Además, hemos visto que los datos obtenidos pueden ser utilizados para conseguir una mayor compresión de otras cuestiones relacionadas con las actividades de los usuarios y los sistemas de ficheros.

En especial hemos comprobado que las gráficas de actividad presentan un potencial muy superior al esperado inicialmente. Estas gráficas exponen patrones de comportamiento muy interesantes para la detección de actividades tanto del usuario como automáticas.

Además, nos hemos dado cuenta que a pesar de que los datos recogidos puedan parecer simples, un buen proceso de limpieza y tratamiento abre las puertas a la observación de resultados de mucho interés.

Respecto a las gráficas de espacio utilizado consideradas en principio fáciles de realizar han resultado un reto. Ha sido necesario filtrar los datos bajo diferentes propiedades y añadir estructuras auxiliares para el seguimiento de la historia de los ficheros.

Respecto de las gráficas de escrituras y lecturas hemos determinado que sería necesario hacer una prueba en un conjunto más amplio de ordenadores/usuarios para cuantificar los límites inferiores y superiores significativos para el número de escrituras y lecturas.

En cuanto a la tabla de importancia hemos elegido un tipo de clasificación que funciona adecuadamente por su simpleza. Al hacer las pruebas hemos considerado otro tipo de heurísticos y nos hemos dado cuenta que cuanto más complejos, más difícil era hacerlos generalistas y no adecuarlos a un set de datos concreto.

Resulta de interés destacar la flexibilidad de nuestro prototipo, por lo que podría ser utilizado en el futuro, para la obtención de otros resultados sólo mediante la inclusión de nuevos módulos que le permitieran ejecutar otros heurísticos. Sin la necesidad de conocer en profundidad la programación del cliente o de la base de datos.

Respecto a la aplicación sobre sistemas de copias de seguridad podemos decir que sacar las listas de ubicaciones de los archivos clasificados por grupos es extremadamente útil ya que se pueden añadir a los filtros de los programas de copias de seguridad directamente.

Finalmente podemos decir que el prototipo funciona adecuadamente para el conjunto de pruebas de control realizadas, estableciendo un punto de partida para desarrollos futuros.

Respecto de la ejecución del trabajo:

Debemos destacar que en la fase de desarrollo del cliente (Programa de recogida de datos) hemos encontrado diferencias significativas en cuanto a lo que la documentación oficial de las librerías deja entender y lo que realmente es posible realizar con ellas. Estas circunstancias fueron de imposible determinación previa dada la dificultad de las pruebas necesarias para ello. Por tanto, hemos tenido que buscar alternativas que nos permitiesen continuar con el desarrollo del trabajo. Este problema ha sido posiblemente el más difícil de solucionar a nivel de implementación ya que existen muy pocas fuentes fiables diferentes a la oficial con las que comparar.

Uno de los elementos más determinantes para el desarrollo del proyecto ha sido la elección de la base de datos. Ésta, estableció una serie de restricciones que modelaron gran parte del trabajo posterior. Nos hemos visto expuestos al lenguaje CQL que, si bien se parece sintácticamente a SQL, no tiene las mismas funcionalidades, siempre restringido a favor de la eficiencia en las consultas.

Para desarrollar el subsistema de análisis, hemos utilizado inicialmente el lenguaje R y posteriormente una adaptación para la librería Pandas de Python. Muchas de las

herramientas utilizadas resultaron nuevas, para las que fue necesaria una fase de inmersión en los conceptos asociados. Ahora consideramos esta inversión temporal, imprescindible y extremadamente valiosa.

La mayor parte del esfuerzo del trabajo se puede dividir en dos apartados importantes. El primero, la comprensión en profundidad del planteamiento realizado y su entorno. Mientras que el segundo ha sido la elección y aprendizaje de las distintas herramientas que podrían utilizarse para el desarrollo del proyecto. Estos dos apartados se han ido retroalimentando a lo largo de las distintas fases del proyecto, provocando siempre un mejor entendimiento de las circunstancias y/o de las herramientas. De esta forma iban surgiendo nuevas preguntas, cada vez más específicas, pero igual de importantes que las anteriores. La resolución de alguno de estos interrogantes ha tenido que quedar fuera del ámbito de este trabajo, sin embargo, pueden ser consideradas como punto de partida para futuros trabajos.

Trabajo futuro:

Como hemos mencionado en párrafos anteriores durante el desarrollo del trabajo han surgido una serie de interrogantes que por la extensión de los mismos han debido quedar fuera del ámbito de este trabajo. Pasamos a mencionar los más relevantes.

Uso de técnicas de “machine learning”, para extender las funcionalidades de nuestro prototipo hemos visto que uno de los aspectos más relevantes es la detección automática de patrones. Debido a la complejidad actual del software y su acelerado ciclo de vida (SDLC), no podríamos pretender detectarlos con patrones estáticos. Será necesario por tanto aplicar técnicas de “machine learning” para la clasificación automática de estos fenómenos.

Asociado al machine learning consideramos que es importante el trabajar con métodos n dimensionales con la intención de detectar relaciones entre los eventos que hasta este momento no conocemos.

Además, sería de interés ver si una implementación del cliente a través de un miniFilterDriver obtiene mejores resultados que los que ya hemos visto.

6. Bibliografía

[1] W. Cazemier, «Backing up Linux and other Unix(-like) systems - Warnings,» 2017. [En línea]. Available: http://www.halfgaar.net/backing-up-unix.

[2] Wikipedia, «Wikipedia - Rsync,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Rsync.

[3] CommVault, «Enterprise data management, protection & backup software,» 2017. [En línea]. Available: https://www.commvault.com/.

[4] EMC, «Avamar. Respaldo y recuperación con deduplicación de datos.,» 2017. [En línea]. Available: https://spain.emc.com/data-protection/avamar.htm.

[5] Veritas technologies, «Backup Exec,» 2017. [En línea]. Available: https://www.veritas.com/product/backup-and-recovery/backup-exec.

[6] L. Cobian, «Cobian Backup,» 2017. [En línea]. Available: http://www.cobiansoft.com/index.htm.

[7] Acronis, «Acronis Small business,» 2017. [En línea]. Available: http://www.acronis.com/es-es/business/enterprise-solutions/small-business/.

[8] Wikipedia, «Wikipedia - Bacula,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Bacula.

[9] Microsoft, «Documentación de .NET,» 2017. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/.

[10] J. Albahari, C# 6.0 in a Nutshell, O'Reilly Media, Inc., 2015.

[11] Microsoft, «Catálogo de referencia y API de Microsoft,» 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/ms123401.aspx.

[12] Docker Inc., «Docker Documentation,» 2017. [En línea]. Available: https://docs.docker.com/.

[13] VmWare Inc, «VmWare Documentation,» 2017. [En línea]. Available: https://www.vmware.com/support/pubs/.

[14] Apache Cassandra, «Apache Cassandra,» 2017. [En línea]. Available: http://cassandra.apache.org/.

[15] R. Bradberry, Practical Cassandra, Boston : Addison- Wesley , 2014.

[16] E. Hewitt, Cassandra The definitive guide, O´Reilly, 2011.

[17] R Development Core Team, «R manuals,» 2017. [En línea]. Available: https://cran.r-project.org/manuals.html.

[18] Python Software Fundation , «Python 2.7.13 documentation,» 2017. [En línea]. Available: https://docs.python.org/2.7/.

[19] F. Romanno, Learning Python, Packt Publishing, 2015.

[20] L. Steven, Python Essentials, Packt Publishing, 2015.

[21] Pandas , «Pandas documentation,» 2017. [En línea]. Available: http://pandas.pydata.org/pandas-docs/stable/.

[22] Django Project, «Django documentation,» 2017. [En línea]. Available: https://docs.djangoproject.com/en/1.11/.

[23] A. Melé, Django By Example, Packt Publishing, 2015.

[24] S. Dauzon, Django Essentials, Packt Publishing, 2014.

[25] Plotly , «Plotly API libraries,» 2017. [En línea]. Available: https://plot.ly/api/.

[26] J. Albahari, Threading in C#, O'reilly Media. Inc , 2011.

[27] Microsoft, «Clase FileSystemWatcher,» Junio 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.io.filesystemwatcher(v=vs.110).aspx.

[28] Microsoft, «Clase EventLog,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.diagnostics.eventlog(v=vs.110).aspx.

[29] Microsoft, «Espacio de nombres System.Threading,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.threading(v=vs.110).aspx.

[30] Microsoft, «Espacio de nombres System.Collections,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.collections(v=vs.110).aspx.

[31] Microsoft, «Clase FileInfo,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.io.fileinfo(v=vs.110).aspx.

[32] Microsoft, «Clase FileSecurity,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.security.accesscontrol.filesecurity(v=vs.110).aspx.

[33] Microsoft, «Clase DirectoryInfo,» 2016. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.io.directoryinfo(v=vs.110).aspx.

[34] Microsoft, «Clase File,» 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.io.file(v=vs.110).aspx.

[35] Microsoft, «Clase Directory,» 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.io.directory(v=vs.110).aspx.

[36] Microsoft, «Espacio de nombres System.Security.Cryptography,» 2017. [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.security.cryptography(v=vs.110).aspx.

[37] J. Carpenter, Data Modeling with Cassandra, 2 ed., O´Reilly, 2016.

[38] S. Sharma, Cassandra designing patterns, Birminghan: Packt Publishing, 2014.

[39] N. Di Tullio, «Docker I: Discovering Docker and Cassandra - Nico's Blog,» junio 2016. [En línea]. Available: http://blog.ditullio.fr/2016/06/10/docker-docker-basics-cassandra/.

[40] DataStax, «Tutorials Datastax Academy,» 2017. [En línea]. Available: https://academy.datastax.com/tutorials.

[41] DataStax, «Casssandra Driver 3.10 documentatión,» 2017. [En línea].