DISEÑO DE UNA HERRAMIENTA DE GESTIÓN DE BENCHMARK Y COMPARACIÓN ENTRE APACHE STORM, APACHE FLINK Y APACHE SPARK

Universidad Carlos III de Madrid

Grado en Ingeniería Informática

Septiembre 23, 2016 Leganés

Autor: Mario Vasile Cabezas

Tutor: José María Álvarez Rodríguez

Título: Diseño de una herramienta de gestión de Benchmark y comparación entre Apache Storm, Apache Flink y Apache Spark.

Autor: Mario Vasile Cabezas
Tutor: José María Álvarez Rodríguez

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Trabajo de fin de grado el día ___ de _____ de 20___ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de _____

VOCAL

SECRETARIO PRESIDENTE

Índice de Contenidos

Contenido

Índice de Figuras

Índice de Tablas

Índice de Gráficas

Agradecimientos

En primer lugar, quiero agradecer a mi familia, en especial a mis padres por todo lo que han hecho por mí, el apoyo incondicional que he recibido su cariño y animo en los momentos más difíciles, a Irene por siempre confiar en mi más que de lo que lo hago yo en mi mismo y apoyarme en los mejores y los peores momentos, gracias por este increíble paseo a tu lado y a mis amigos por los consejos y las palabras de aliento, en especial a Daniel Cano.

En segundo lugar, quiero dar las gracias a José María, por haberme permitido realizar este Trabajo Fin de Grado el cual ha sido una experiencia de aprendizaje y superación continua. También quiero aprovechar para agradecer su ayuda en el desarrollo del mismo.

En tercer lugar, quiero dar las gracias a todos los excelentes profesores que me en encontrado en vida de estudiante y que han ayudado a llegar al punto en que me encuentro ahora, tenemos grandes profesores, excelentes profesionales y maravillosas personas.

Y en cuarto y último lugar, quiero agradecer y dedicar este presente Trabajo Fin de Grado, a la persona más importantes de mi vida, gracias a ella soy una persona trabajadora y luchadora. Nunca existirán palabras para expresarte mi admiración y cariño, estés donde estés abuela, muchas gracias por todo.

Gracias a todos ellos hoy estoy realizando este Trabajo Fin de Grado.

Resumen

Nos encontramos en un momento, donde la informática es capaz de procesar cantidades enormes de información que son generadas día a día, por los sistemas de medición de tráfico de las ciudades, evolución de la temperatura a lo largo de día …

Hasta hace poco tiempo, lo único que se hacía con todos estos datos que se generaban era almacenarlos, sobre todo bancos y grandes compañías, pues el almacenamiento de la información era muy barato.

En 2003 se empieza a ver trabajos para analizar la gran cantidad de información que se genera. En nuestro caso particular nos desplazamos hasta el año 2011/2012, donde se libera las primeras versiones alfa de Apache **STORM**, Apache **SPARK** y algo más tarde 2015 Apache **FLINK**.

Estas herramientas crean una capa que permite abstraernos de paralelizar el código, distribuirlo en clúster etc. De esto, se encargan estas herramientas, permitiendo desarrollar aplicaciones más complejas con menos código y más rápido.

Cada una de las herramientas nos permiten realizar análisis de grandes cantidades de datos, generando modelos de aprendizaje automático, o analizando los datos en tiempo real, como puede ser el cambio y evolución en twitter de una noticia.

Es este Trabajo Fin de Grado se pretende realizar un Benchmark o herramienta de comparación, que permita comparar cada uno de los Frameworks, con el fin de determinar en qué escenarios son más recomendados.

La idea general, es enfrentar un mismo algoritmo en las distintas herramientas mientras observamos tres variables para medir el rendimiento:

I. **Tiempo** que tarda en completarse la tarea.
II. Memoria **RAM** que se ha empleado durante toda la ejecución.
III. Uso de **CPU** que ha utilizado para completar la tarea.

Para tratar que las pruebas se realicen en entornos limpios y sean lo más asépticas posibles, cada uno de los Frameworks se montaran sobre contenedores Docker, con el

objetivo que cada vez que se realice una prueba sea la primera vez que se utiliza el entorno y la facilidad para que tenemos para moverlos a otros sistemas.

Abstract

We are at a point where the computer is able to process huge amounts of information that are generated daily by the traffic measurement systems of cities, changing the emperature throughout the day ...

Until recently, all that was done with all these data generated was stored, especially banks and large companies, as the storage of information was very cheap.

In 2003 he begins to see work to analyze the vast amount of information generated. In our particular case we move through the year 2011/2012, where the first alpha versions of Apache STORM, Apache SPARK is released and somewhat later (2015) Apache FLINK.

These tools create a layer that allows abstract from parallelize the code, distribute clustered etc. This, these tools are responsible, allowing to develop more complex applications with less code and faster.

It is this Bachelor´s degree, we aim to make a benchmark or comparison tool that allows to compare each of the frameworks, in order to determine which scenarios are most recommended.

The general idea is to face the same algorithm on different tools while observing three variables to measure performance:

I. **TIME** it takes to complete the task.
II. **RAM** that has been used throughout execution.
III. **CPU** used to complete the task.

To treat the tests are carried out in clean environments and are as aseptic as possible, each of the Frameworks were mounted on Docker containers, with the aim that every time a test is performed is the first time the environment is used and ease we have to move to other systems.

Glosario[1]

Algoritmo

Conjunto ordenado, sistemático y finito de operaciones que permite hallar la solución a un problema.

Apache Flink

Framework de código abierto impulsado por la comunidad para la analítica de grandes cantidades de datos distribuidos.

Apache Spark

Potente motor de procesamiento de código abierto construido en torno a la velocidad, la facilidad de uso y análisis sofisticados. Originalmente fue desarrollado en la Universidad de Berkeley en 2009.

Apache Storm

Sistema de procesamiento en tiempo real distribuido que le permite procesar gran cantidad de datos.

Apache2

Servidor web HTTP de código abierto.

API

Interfaz de Programación de Aplicaciones, conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Back-end

Parte que procesa la entrada desde el front-end, panel de administración y gestión.

[1] Las definiciones de los términos han sido obtenidas de Wikipedia, RAE o Definición Personal.

Batch

Procesamiento por lotes

BBDD

Base de datos, conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Benchmark

"Comparativa", técnica utilizada para medir el rendimiento de un sistema o componente del mismo.

Big Data

Concepto que hace referencia al almacenamiento de grandes cantidades de datos y a los procedimientos usados para encontrar patrones repetitivos dentro de esos datos.
Bundle

Empaquetado para almacenar en el propio archivo las plantillas, archivos, imágenes asociados a un proyecto.

Clusters

Conjuntos o conglomerados de computadoras construidos mediante la utilización de hardware comunes y que se comportan como si fuesen una única computadora.

CPU

Hardware dentro de una computadora u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

CSS

Hoja de estilo en cascada, lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML.

CSV

Archivos con valores separados por comas.

Curl

Librería de funciones para realizar acciones sobre archivos que hay en URLs de Internet, soportando los protocolos más comunes, como HTTP

Dashboard

Interfaz que permite al usuario puede administrar el equipo y/o software.

Docker

Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Unix.

Driver

Programa que permite al sistema interaccionar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz (posiblemente estandarizada) para utilizar el dispositivo.

Exabyte (EB)

Unidad de medida de almacenamiento de Datos cuyo símbolo es el 'EB'. Equivale a 10^18 bytes.

Fork

Creación de un proyecto en una dirección distinta de la principal u oficial tomando el código fuente del proyecto ya existente.

Framework

Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Front-end

En diseño de software el front-end es la parte del software que interactúa con el o los usuarios.

GFS

Sistema de archivos distribuido propietario desarrollado por Google, que soporta toda su infraestructura informática de procesamiento de información en nube. Está especialmente diseñado para proveer eficiencia, fiabilidad de acceso a datos usando sistemas masivos de cluster de procesamiento en paralelo.

Gigabyte (GB)

Unidad de medida de almacenamiento de Datos cuyo símbolo es el 'GB'. Equivale a 10^9 bytes.

Git

Software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Github

Plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

Hadoop

Framework de software que soporta aplicaciones distribuidas bajo una licencia libre. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos. Hadoop se inspiró en los documentos Google para MapReduce y Google File System (GFS).

Hardware

Partes físicas tangibles de un sistema informático. Componentes eléctricos, electrónicos, electromecánicos, mecánicos…

HTML

Lenguaje de marcas de hipertexto, hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones.

IDE

Entorno de **D**esarrollo **I**ntegrado, aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Internet Of Things (IoT)

Internet de las cosas (en inglés, Internet of things, abreviado IoT) es un concepto que se refiere a la interconexión digital de objetos cotidianos con internet.

JS

JavaScript (JS), lenguaje de programación interpretado, dialecto del estándar ECMAScript. Basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Kernel

Software que constituye una parte fundamental del sistema operativo.

Latencia

Tiempo que transcurre entre un estímulo y la respuesta que produce.

Linux

Sistema operativo libre, basado en Unix. Es uno de los principales ejemplos de software libre y de código abierto.

Machine Learning

Rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos.

MapReduce

Modelo de programación utilizado por Google para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras. Documentación Oficial: http://research.google.com/archive/mapreduce.html

Maven

Herramienta de software para la gestión y construcción de proyectos Java.

Memoria RAM

Memoria de acceso aleatorio (Random Access Memory, RAM) se utiliza como memoria de trabajo de computadoras para el sistema operativo, los programas y la mayor parte del software. En la RAM se cargan todas las instrucciones que ejecuta la unidad central de procesamiento (procesador) y otras unidades del computador.

Microservicios

Método conforma una aplicación o herramienta mediante la conjunción de diversos servicios independientes que se despliegan según se vayan necesitando.

Modelo Entidad-relación

Herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información, así como sus interrelaciones y propiedades.

Mysql

Sistema de gestión de bases de datos relacional desarrollado. Está considerada como la base datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

ORM

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional.

Petabyte (PB)

Unidad de medida de almacenamiento de Datos cuyo símbolo es el 'PB'. Equivale a 10^15 bytes.

PHP

Lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

PHPmyAdmin

Herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando el navegador.

POO

Programación orientada a objetos, paradigma de programación. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

RDD

Estructura es una colección de objetos inmutables y distribuidos, donde cada RDD es dividido en particiones lógicas, las cuales pueden ser procesadas por diferentes nodos del cluster.

Repositorio

Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

RestFul

Conjunto de protocolos abiertos y estándares utilizados para el intercambio de datos entre aplicaciones o sistemas con el uso de HTTP.

Root

En sistemas operativos del tipo Unix, root (o superusuario) es el nombre convencional de la cuenta de usuario que posee todos los derechos en todos los modos.

S.O.

Sistema Operativo.

Script

Archivo de órdenes para automatizar procesos.

Smart Cities

Ciudades que se sirven de infraestructuras, innovación y tecnología con el fin de mejor calidad de vida y conseguir una gestión prudente de los recursos naturales.

Software

Programas informáticos que hacen posible la realización de tareas específicas dentro de un computador.

SSD

Unidad de estado sólido, del acrónimo inglés de Solid-State Drive, es un tipo de dispositivo de almacenamiento de datos que utiliza memoria no volátil, como la memoria flash, para almacenar datos, en lugar de los platos o discos magnéticos de las unidades de discos duros (HDD).

Stack

Conjuntos de distintas tecnologías que se unen con el fin de crear un estándar de facto.

Stack Lamp

Acrónimo usado para describir un sistema de infraestructura de internet que usa: **L**inux, **A**pache, **M**ySQL y **P**HP

Streams

Flujo de datos.

Symfony

Framework diseñado para optimizar el desarrollo de las aplicaciones web.

Terabytes (TB)

Unidad de medida de almacenamiento de Datos cuyo símbolo es el 'TB'. Equivale a 10^12 bytes.

Throughput

Volumen de trabajo o de información neto que fluye a través de un sistema.

Topología

Mapa lógico de una red Storm para el procesamiento de datos.

Twitter

Red social, servicio de microblogging.

Ubuntu

Sistema operativo basado en GNU/Linux y que se distribuye como software libre.

Unix

Sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T.

Vagrant

Herramienta para la creación y configuración de entornos de desarrollo virtualizados.

Virtualbox

Software de virtualización, por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como sistemas invitados dentro de otro sistema operativo anfitrión, cada uno con su propio ambiente virtual.

Vmware

Software de virtualización, por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como sistemas invitados dentro de otro sistema operativo anfitrión, cada uno con su propio ambiente virtual.

World Wide Web

Sistema de distribución de documentos de hipertexto o hipermedios interconectados y accesibles vía Internet. Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de esas páginas usando hiperenlaces.

CAPÍTULO 1

1 Introducción

1.1 Motivación

Desde hace mucho tiempo, generamos una gran cantidad de información que cada vez es complicado analizar. En 1880 ya empiezan a tener problemas para calcular el censo de Estados Unidos, pues se necesitaron 8 años en procesar toda la información. Esto provocó que se empezara a buscar mejores formas de procesar los datos que recababan.

Avanzando en el tiempo hasta 1941, se empieza a hablar de la explosión de la información haciendo referencia a la dificultad que suponía gestionar tantos datos.

En 1970 **Edgar F. Codd** publica un artículo donde habla del modelo relacional de base de datos, esto permitirá acceder a la información de forma mucho más rápida y reduciendo la complejidad que existía hasta ese momento.

Hoy en día, las transacciones de datos rutinarias, acceder a cuentas bancarias, utilizar tarjetas de crédito, reservas de hotel, realizar compras a través de Internet utilizan estructuras basadas en la teoría de la base de datos relacional.

A medida que pasaba el tiempo, la velocidad con la que se creaba información aumentaba. Los avances tecnológicos permiten a las empresas empezar a trabajar con los datos que almacenaban para tomas decisiones de negocio, lo que se conoce hoy en día como **Business Inteligence**.

Pero no fue hasta que en la década de los 90 aparece World Wide Web y el desarrollo tecnológico se dispara, así como la información que se genera que empezara a apilarse.

Peter Lyman y Hal R. Varian de Universidad de Berkeley publicaron el primer estudio que cuantificaba, en términos de almacenamiento informático, la cantidad total de

información nueva y original creada en el mundo al año. El estudio, titulado *How Much Information?*, se completó en 1999, un año en el que el mundo produjo unos 1,5 exabytes (EB) de información.

1 EB = 1 073 741 824 GB = 10^9 GB

En 2006 se crea **Hadoop**, permitiendo trabajar con petabytes de datos (**1 PB = 1048576 GB**). Para esta titánica tarra, hizo uso de MapReduce y GFS (Google File System) de Google como base para su desarrollo. Esto permitió el procesamiento en paralelo distribuido de enormes cantidades de información [1].

A finales de 2011 se empieza a desarrollar Apache Storm, a mediados de 2012 Apache Spark y por último en junio de 2015 aparece Apache Flink. Estos Frameworks, pretenden ser una mejora de Hadoop y de esta forma poder analizar y procesar más información en menor tiempo y de forma más eficiente.

Nos encontramos en un momento en el cual tenemos la capacidad de almacenar y procesar la información que se genera con el uso de las tecnologías que usamos. Con la llegada del IOT del inglés Internet Of Things (Internet **de las cosas**) o las Smart Cities (Ciudades **Inteligentes**) muchos más datos se van a generar y más importante es poder procesar la información en tiempo real y poder tomar decisiones en consecuencia.

Para una empresa poder analizar los cientos de Terabytes de datos que han generado y poder inferir conocimiento para mejorar sus sistemas de producción y optimizar recursos es algo que cualquier empresa quiere y necesitan conocer.

1.2 Objetivos

El principal objetivo de este Trabajo Fin de Grado será:

I. Diseñar un Benchmark que permita automatizar las pruebas y medir el rendimiento de cada uno de las herramientas.
II. Comprobar cómo se comporta cada herramienta en los distintos escenarios a los que se enfrente.
III. Comprobar el rendimiento de un mismo algoritmo en las distintas herramientas.

1.3 Entorno Socio-Económico

Las grandes empresas de la actualidad como **Google, Facebook, Twitter, Microsoft, Yahoo!, BBVA, Telefónica** … Se encuentran invirtiendo tiempo, dinero y esfuerzo en ser capaces de predecir eventos, tomar mejores decisiones para el negocio, y todo ello a causa de la inmensa cantidad de datos que tienen almacenados y que generan.

Algunas estrategias basadas en el análisis de datos: [2]

- Descubrir y alcanzar nichos de mercado a los que antes no podíamos llegar.
- Aumentar ventas a través de sistemas de recomendación y venta cruzada.
- Ofrecer al cliente una experiencia de consumo personalizada.
- Anticipar y prevenir fugas de clientes.
- Determinar el potencial de cada cliente / segmento.
- Predecir las necesidades de nuestros clientes.

Hoy en día está muy de moda oír hablar sobre **la privacidad de las personas en internet** y cómo las grandes compañías "pelean" para disponer de una posición ventajosa. Recientemente hemos oído hablar como la famosa aplicación de mensajería **WhatsApp** iba a compartir tu número de teléfono en Facebook [3], pero, ¿Por qué esto tiene que ver con el análisis de Big Data? Analizando de forma masiva todos nuestros datos, las empresas son capaces de vendernos productos de forma más eficiente. *Lo que podríamos resumir como vender más gastando menos.*

Otro punto importante que nos afecta es la **lucha contra el crimen** [4], antes si se tenía una prueba era necesario ir cotejando uno por uno con las muestras existentes. Hoy podemos hacer uso del potencial de las herramientas de Big Data para procesar grandes cantidades de datos en muy poco tiempo, esto permite que las investigaciones sean más rápidas y eficientes.

Por estos motivos cada día estas herramientas tienen más impacto en nuestra vida cotidiana a pesar de que no nos damos cuenta, tanto social como económicamente.

1.4 Marco Regulador

En este apartado repasaremos las medidas legales que debemos de tener en consideración a la hora de implantar e utilizar el sistema que se propone en este Trabajo Fin de Grado.
Puesto que la aplicación puede procesar y analizar información de carácter personal, es necesario cumplir con la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal, en adelante **LOPD** [5], haciendo mención al **Artículo 4 Calidad de los datos**, por el cual los datos personales deberán ser:

- Adecuados
- Pertinentes
- No Excesivos

Los datos de carácter personal objeto de tratamiento no podrán usarse para finalidades incompatibles con aquellas para las que los datos hubieran sido recogidos. No se considerará incompatible el tratamiento posterior de éstos con fines históricos, estadísticos o científicos.

También tendremos en cuenta el **Artículo 7, Datos especialmente Protegi**dos de la LOPD, por la cual:

- No se crearán ficheros que contengan datos de carácter personal como ideología, religión, creencias, origen racial, étnico o vida sexual.

- En caso de que se almacene datos de carácter personal como ideología, religión, creencias, origen racial, étnico o vida sexual, se pedirá al usuario que acepte unos términos de uso que permitan a la empresa la utilización de dichos datos.

Toda utilización se software, está sujeto a unas licencias las cuales hay que analizar y saber cuáles son nuestras obligaciones y deberes tenemos pues según el **Artículo 10 Obras y títulos originales** de, Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia, **en adelante LPI** [6], establece que:

"Son objeto de propiedad intelectual todas las creaciones originales literarias, artísticas o científicas expresadas por cualquier medio o soporte, tangible o intangible, actualmente conocido o que se invente en el futuro, comprendiéndose entre ellas: **Los programas de ordenador.**"

Y atendiendo al **Artículo 11 Obras derivadas** de la LPI, Sin perjuicio de los derechos de autor sobre la obra original, también son objeto de propiedad intelectual:

I. Las traducciones y adaptaciones.
II. Las revisiones, actualizaciones y anotaciones.
III. Los compendios, resúmenes y extractos.
IV. Los arreglos musicales.
V. Cualesquiera transformaciones de una obra literaria, artística o científica.

Por lo que al modificar un programa con derechos de autor no elimina los derechos que existen sobre dicho software. Por último, el **Artículo 158 Función de salvaguarda de los derechos en el entorno digital** de la LPI, establece que en caso de incumplir cualquiera de los derechos de autor, conllevará una multa comprendida entre **150.001** y **600.000** euros.

1.5 Estructura del Documento

Este documento contiene seis capítulos. A continuación, Una breve introducción de cada uno de ellos.

- **Capítulo 1: Introducción.** Expone la motivación y objetivos de la tesis, estructura del documento y la estructura de la memoria.
- **Capítulo 2: Estado del Arte.** Realiza una pequeña introducción al concepto de BIG DATA y un análisis de los Frameworks analizados
- **Capítulo 3: Descripción General del Sistema.** Explica las herramientas utilizadas y el porqué de la elección en este proyecto.
- **Capítulo 4: Evaluación y Resultados.** Evaluación del sistema desarrollado y los resultados obtenidos.
- **Capítulo 5: Planificación y presupuesto.** En este apartado explicamos las fases en que se ha planificado el proyecto, tareas realizadas y el presupuesto necesario para llevar a cabo
- **Capítulo 6: Trabajos futuros.** Expone las propuestas de posibles desarrollos futuros.

CAPÍTULO 2

2 Estado del Arte

Para realizar el presente Trabajo de Fin de Grado, se ha realizado un entendimiento y comprensión de lo que es el procesamiento de BIG DATA.

Por ello, lo primero que se va a realizar es una introducción al procesamiento de BIG DATA, seguido de una explicación breve de algunos de los algoritmos que se usan en los tres Framework que vamos a comparar y que se han utilizado en este Trabajo.

2.1 Introducción

A continuación, vamos a realizar una comparativa con distintas soluciones que existen en el mercado con el fin de demostrar que no resuelven el problema por completo.

La primera solución que vamos a analizar, es un Benchmark desarrollado por **Yahoo!** La solución es pública y está alojada en GitHub [7]

Estas son algunas de las carencias que hemos detectado:

- Esta solución **NO** permite comparar la API completa de las 3 herramientas.
- La única medida de rendimiento que nos ofrece, es el tiempo en completar los algoritmos, pero no nos ofrece una visión del consumo de memoria RAM/CPU.
- Para poder ejecutar la prueba, sería necesario instalar los Frameworks en local, lo que reduce la velocidad de portabilidad del Benchmark.
- No permite almacenar un histórico de pruebas realizadas para hacer una comparación del mismo algoritmo con distintos ficheros.

La segunda solución que vamos a estudiar, es una Benchmark desarrollado por **Intel**. La solución es pública y está alojada en GitHub [8]

Estas son algunas de las carencias que hemos detectado:

- Únicamente dispone analiza Apache Storm.
- Los resultados del test son generados en un archivo CSV y no son graficados.
- No permite almacenar un histórico de pruebas realizadas para hacer una comparación del mismo algoritmo con distintos ficheros.
- Para poder ejecutar la prueba, sería necesario instalar los Frameworks en local, lo que reduce la velocidad de portabilidad del Benchmark.

NOTA*: *Debemos destacar la escasa variedad de herramientas, que nos permitan realizar comparaciones con estos tres Frameworks.*

2.2 Comparativa de Herramientas

CARACTEÍSTICAS	Soporta los 3 Frameworks	Permite toda la API	Frameworks aislados del S.O.	Generan Gráficas	Miden Tiempo	Miden RAM %	Miden CPU %
YAHOO!	✓	✗	✗	✗	✓	✗	✗
intel	✗	✓	✗	✗	✓	✗	✗
Propuesta	✓	✓	✓	✓	✓	✓	✓

Tabla I Comparación de Herramientas Benchmark

Como se puede observar en la tabla anterior, ninguno de las soluciones analizadas cumple con todos los requisitos que buscamos satisfacer y por este motivo se decide realizar este proyecto.

2.3 Frameworks de Análisis de BIG DATA

En este apartado vamos a realizar una introducción a los 3 Frameworks que vamos a comparar para ver cuáles son las características coincidentes y cuáles los diferencian.

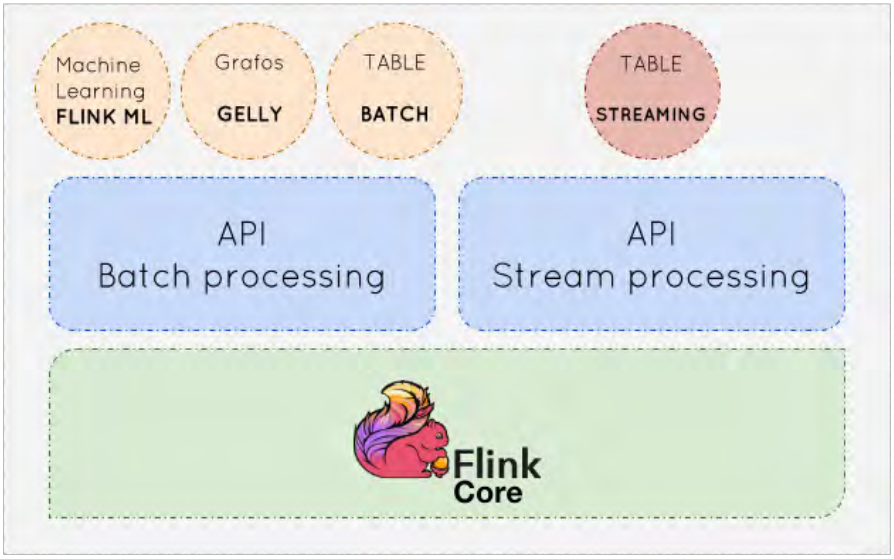
2.3.1. Apache Flink

Empezaremos hablando de Apache Flink [9], este Framework de procesamiento en **streams** o en castellano **flujo de datos,** apareciendo por primera vez el agosto del 2014 siendo un fork o bifurcación de Stratosphere. Este proyecto es europeo en concreto fue fundado por la *German Research Foundation*.

Ilustración I Logo de Apache Flink

Flink nos proporciona capacidad de distribución de datos, teniendo como principal novedad frente a las herramientas ya instauradas como Hadoop [10], es la **NO** utilización del modelo de programación ideado por **Google** *MapReduce* [11] el cual permite computación paralela sobre grandes cantidades de datos.

Apache Flink se diseña desde un inicio buscando una alternativa al modelo desarrollado por Google, pero disponemos de la librería y estructuras necesarias dentro del framework si deseamos utilizar este modelo en el desarrollo de nuestra aplicación



[2]Ilustración II Vista de los Componentes. Apache Flink

En la imagen anterior podemos ver cuáles son los componentes del Framework, partiendo de flink-core que se encuentra en la base del gráfico y donde se encuentran todas las utilidades básicas de la herramienta.

Seguidamente tenemos dos API una de ellas para el procesamiento en Batch y otra y más representativa de Flink, la API para el procesamiento en streaming. Además, contamos con ciertas librerías para el procesamiento de grafos.

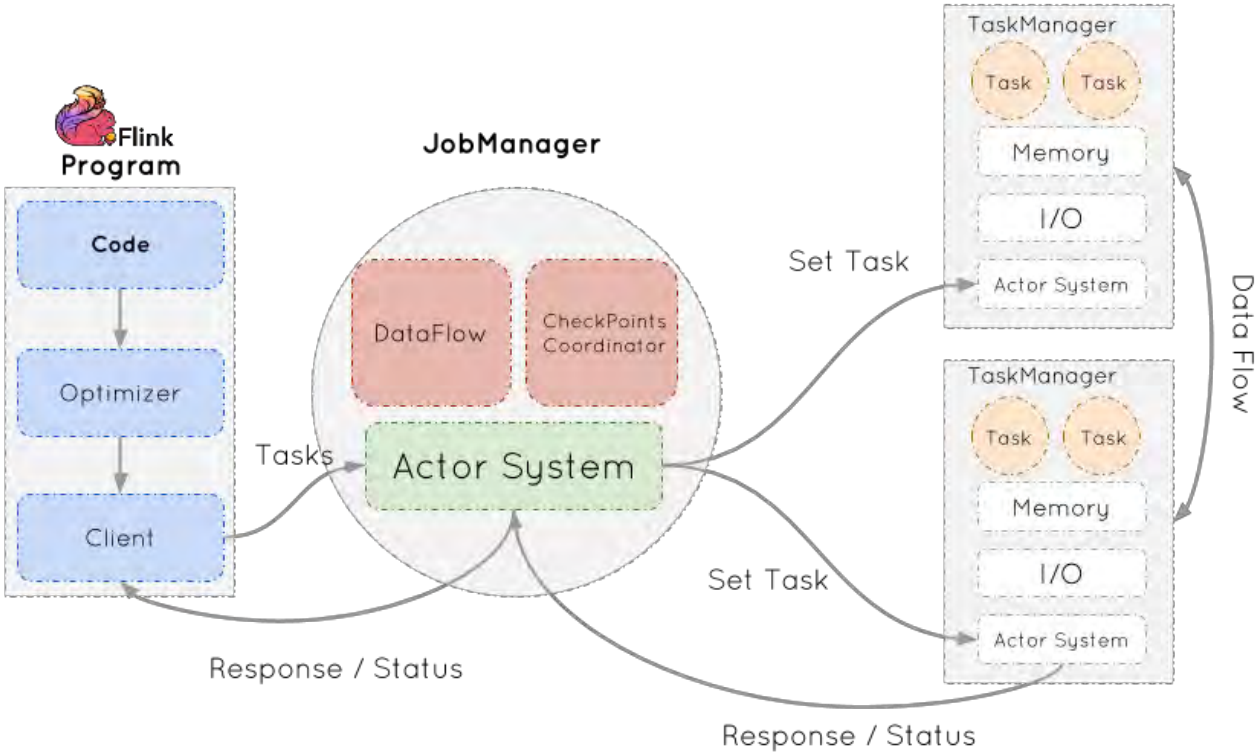
[2] Propiedad Imagen: https://www.adictosaltrabajo.com/tutoriales/introduccion-a-apache-flink/

2.3.1.1. Arquitectura

La arquitectura básica del Frameworks está compuesta principalmente por dos componentes:

- JobManager
- TaskManager

La función del JobManager es coordinar los trabajos que se van a lanzar en el sistema partiendo en tareas más pequeñas que se ejecutarán en paralelo, de las cuales se encarga en TaskManager, en la siguiente imagen se muestra una visión de cómo funciona la herramienta.



[3]Ilustración III Arquitectura de Apache Flink

[3] Propiedad de la imagen: https://www.adictosaltrabajo.com/tutoriales/introduccion-a-apache-flink/

2.3.2. Apache Spark

Ahora analizaremos Apache Spark [12], este Frameworks de procesamiento en batch o en castellano procesamiento por lotes, fue lanzando por primera vez en junio de 2012. Este Frameworks es norteamericano y está impulsado por la **universidad Berkeley de California.**

Ilustración IV Logo Apache Spark

Spark es un Frameworks de computación en cluster el cual implementa una nueva estructura de datos llamada **RDD** [13] (Resilient Distributed Datasets). Esta estructura es una colección de objetos inmutables y distribuidos, donde cada RDD es dividido en particiones lógicas, las cuales pueden ser procesadas por diferentes nodos del clúster. Esto permite que el uso de MapReduce junto con RDD consiga que las operaciones sean más rápidas y eficientes.

[4]Ilustración V Vista de los Componentes. Apache Spark

[4] Propiedad de la imagen: https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

En la imagen anterior podemos observar todas las librerías que soporta Apache Spark por defecto. Como vemos todas ellas descansa sobre **sparkContext**, que se encuentra en la base de la imagen y contiene las herramientas básicas, por encima, descansas las siguientes librerías:

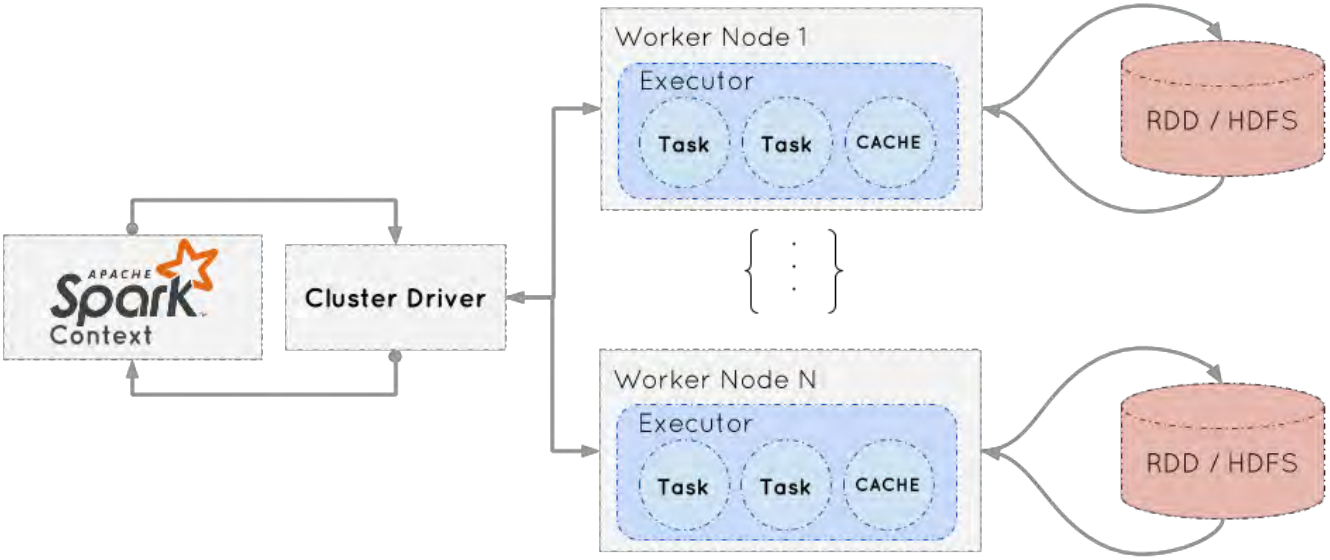
- Spark SQL que permite trabajar con BIG DATA teniendo una sintaxis similar a SQL.
- MLib dispone de algoritmos de Machine Learning o Aprendizaje Automático como puede ser K-means, Bayes …
- GraphX el cual nos permite generar / recorrer grafos usando computación paralela.
- Spark Streaming: Esta librería lo que pretende cubrir es una característica para la que no se diseñó en un primer momento el Frameworks pero que la evolución del mercado, ha obligado a introducirlas.

2.3.2.1. Arquitectura

Vamos a analizar cómo funciona la arquitectura de este Frameworks, para ello debemos hablar del *SparkContext*, el cual tendrá la configuración con la que se va a ejecutar la aplicación como pueden ser, el **número de núcleos** a utilizar, la **memoria disponible** para el driver o para el ejecutor. El resto de parámetros se puede ver en el siguiente enlace la documentación del framework

El siguiente concepto, sería el de ejecutor o **Executer** que son los encargados de realizar el procesamiento de la información en los nodos existentes y dichos ejecutores se encargan de procesar los fragmentos de código que se les ha asignado, siendo denominado estos fragmentos como tareas o **Task**.

En la siguiente imagen se puede ver una imagen con que describe la arquitectura del Frameworks en modo Cluster.



[5]Ilustración VI Arquitectura de Apache Spark

[5] Propiedad de la imagen: https://www.adictosaltrabajo.com/tutoriales/introduccion-a-apache-spark-batch-y-streaming/

2.3.3. Apache Storm

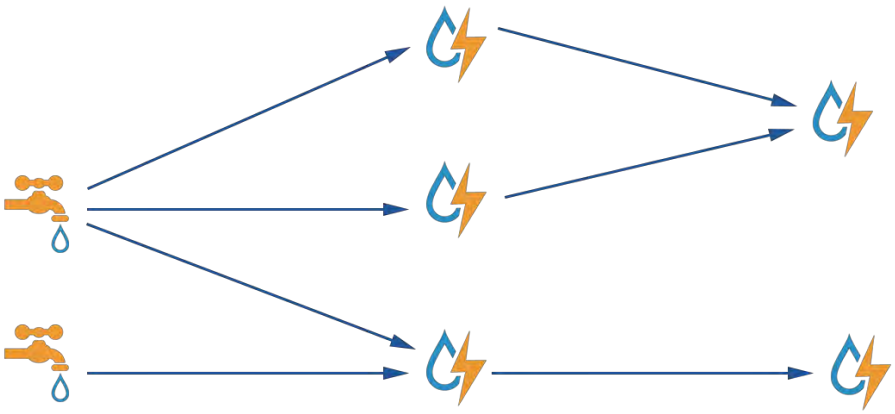
Ilustración VII Logo Apache Storm

Como tercer Frameworks de procesamiento de BIG DATA analizaremos Apache **Storm**. Esta herramienta realiza procesamiento de **stream** al igual que Apache Flink. La primera versión aparece en septiembre de 2011 y posteriormente fue adquirida por **Twitter**.

Apache Storm nos permite estar procesando **stream** de datos en tiempo real desde varios orígenes o fuentes con las siguientes características:

- Procesamiento distribuido.
- Tolerante a fallos.
- Alta disponibilidad.

En la siguiente imagen vemos una abstracción de **alto nivel** de cómo funciona el procesamiento en Storm.



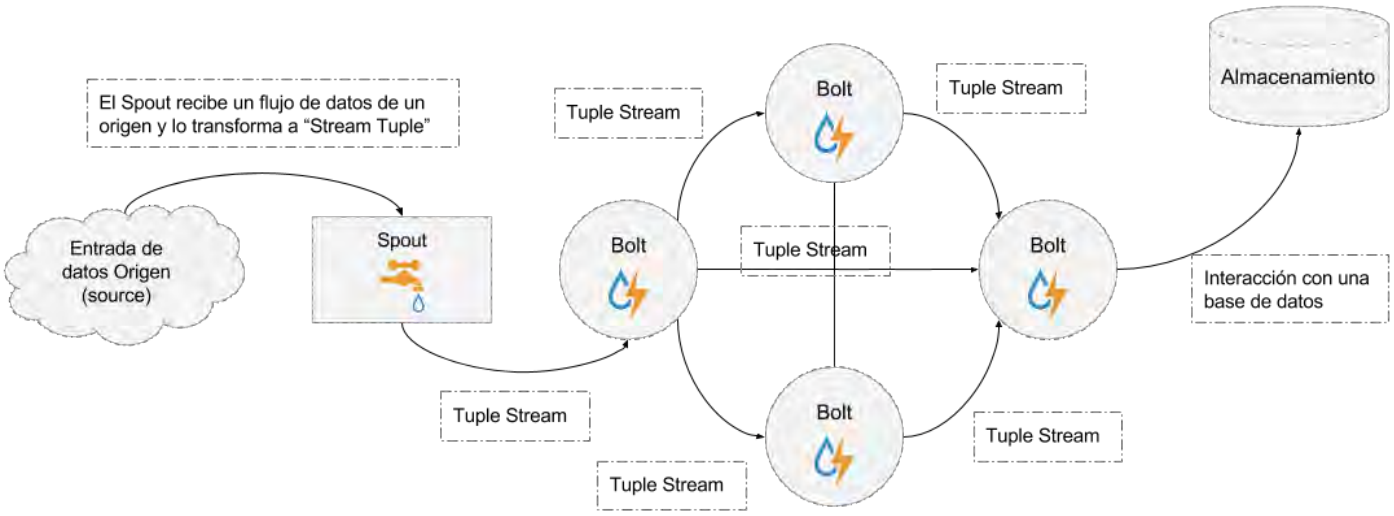
[6]Ilustración VIII Abstracción de cómo funciona Apache Storm

[6] Propiedad de la imagen: http://storm.apache.org/

Los dos principales componentes de Storm son [14]:

I. **Tuple** o Tuplas: Esta es la estructura de datos principal en Storm. Consiste en una lista ordenada de elementos, generalmente se modela como un conjunto de datos separando los valores por comas.
II. **Stream** o Flujo: Secuencia desordenadas de Tuple
III. **Spouts**: Este componente se encarga de recoger el flujo de los datos de entrada que pueden ser de diversos sitios como:
    a. Twitter Streaming API
    b. Apache Kafka
    c. etc.
IV. **Bolts**: Este componente es el encargado de procesar o transformar la información que recibe y genera un nuevo stream de salida. Dentro de este componente se pueden realizar operaciones de Filtrado, Agregación, uniones o interacciones con BBDD etc.

Vamos a proceder a analizar un ejemplo de funcionamiento:



[7]Ilustración IX Funcionamiento de Apache Storm

Como se puede observar en la imagen anterior podemos distribuir operaciones en varios Bolts, y finalmente podemos almacenarlos en una BBDD como se muestra en el ejemplo.

[7] Propiedad de la imagen: http://www.tutorialspoint.com/apache_storm/apache_core_concepts.htm

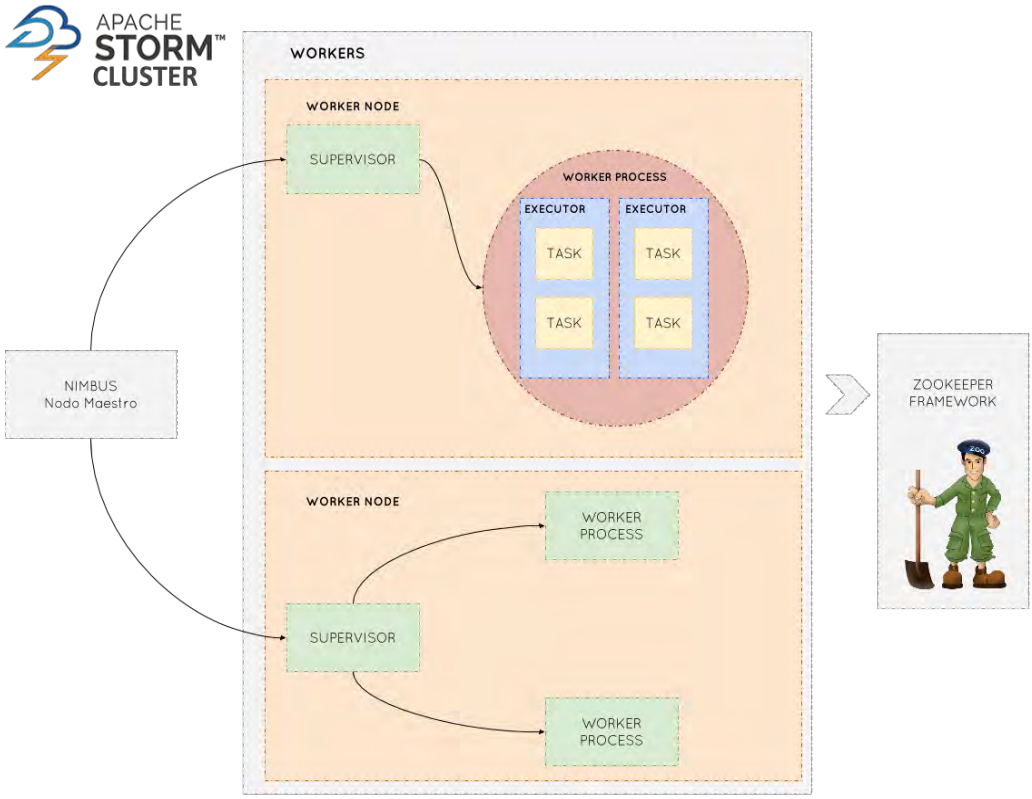
Si seguimos analizando el ejemplo, las uniones que existen entre el Spout y el Bolt, se denomina **Topology o Topología** y se representan como un grafo dirigido, donde los vértices representan la computación de los datos y los bordes son los stream de datos.

El siguiente paso que daremos será el de procesar la topología, Storm utiliza el concepto de **Task o Tarea**, que consiste en la ejecución de todos y cada uno de Spout y Bolt por Storm donde pueden tener múltiples instancias ejecutándose en múltiples hilos separados.

Existe la entidad de **Worker**, que se encarga de estas escuchando los nodos donde se ejecutan las tareas por los trabajos y se encarga de detenerlos o iniciarlos en función de los trabajos nuevos que lleguen, en el siguiente apartado se profundiza en él.

2.3.3.1. Arquitectura

En este apartado vamos a proceder a analizar la arquitectura. [15]



[8]Ilustración X Arquitectura de Apache Storm

[8] Propiedad de la imagen: https://www.tutorialspoint.com/apache_storm/apache_storm_cluster_architecture.htm

En la anterior imagen, podemos ver cómo se relacionan los distintos componentes que forman la arquitectura de la aplicación. A continuación, procedemos a describirlos.

2.3.3.1.1. Nimbus

Nodo **maestro** del cluster en Storm, todos los demás nodos dentro del clúster, son denominados nodos **Worker.** El nodo maestro es el encargado de distribuir los datos entre todos los nodos Worker, asigna tareas a los nodos Worker y supervisar los fallos.

2.3.3.1.2. Supervisor

Este nodo sigue las instrucciones que recibe de Nimbus. Un supervisor tiene múltiples **procesos Workers** y guía los procesos Worker para completar las tareas asignadas por Nimbus.

2.3.3.1.3. Worker Process

Será el encargado de ejecutar **Task** o tareas en relación a una Topology o Topología específica. Por sí mismo no ejecuta tasks, en su lugar crea **Executors** o ejecutores y les pide que realicen una tarea en particular. Un proceso Worker tendrá múltiples **Executors.**

2.3.3.1.4. Executor

Un executor es un solo hilo que carga y ejecuta un nuevo proceso hijo por un proceso Worker. Un ejecutor puede ejecutar uno o más Tasks, pero únicamente del mismo spout o bolt.

2.3.3.1.5. Task

Es un spout o bolt, realiza el procesamiento de los datos.

2.3.3.1.6. ZooKeeper Framework

Servicio usado por el cluster, para coordinar entre los nodos y el mantenimiento de los datos compartidos usando técnicas de sincronización robustas. Puesto que Nimbus es sin estado, ZooKeeper es el responsable de supervisar el estado de los nodos de trabajo.

ZooKeeper ayuda al supervisor para que interactúe con Nimbus y es el responsable de mantener el estado entre Nimbus y el supervisor.

Storm por naturaleza es sin estado, esta arquitectura tiene inconvenientes, pero a la vez es lo que permite a Storm procesar los datos en tiempo real de la mejor manera y más rápida posible.

2.4 Resumen de los Tres Frameworks

En la siguiente tabla comparativa podemos ver las características de cada Frameworks.

CARACTERÍSTICAS	Flink	Spark	Storm
Machine Learning	FlinkML	MLib	trident-ml (No Nativa)
Grafos	Gelly	GraphX	-
Soporte SQL	Table API	Spark SQL	Storm SQL
Procesamiento	stream processing	micro-batching	stream processing
Lenguajes	Java, Scala	Scala, R, Java, Python	Java
API	Alto nivel	Alto nivel	Bajo Nivel
Latencia	Bajo	Medio	Bajo
Throughput	Alto	Alto	Medio

Tabla II Comparativa entre los tres Frameworks

Después de realizar el análisis de los tres Frameworks vamos a realizar una pequeña recapitulación de cada uno de ellos:

- Spark y Flink disponen de librerías nativas para el procesamiento de Batch y de Stream y Storm está más especializado en procesamiento en Stream.
- Spark y Flink los podríamos categorizar como de "alto nivel", pues tienen una curva de aprendizaje menor, mientras que Storm es más complicado de usar, pero te permite afinar más cómo quieres realizar el procesamiento de los datos.

CAPÍTULO 3

3 Descripción General del sistema

3.1 Introducción

Para el desarrollo del presente trabajo hemos utilizado diferentes tecnologías uniéndolas entre ellas para lograr el objetivo propuesto, a continuación, indicamos cuales son.

Hemos utilizado Apache *Storm*, Apache *Spark* y Apache *Flink* como herramientas sobre las cuales vamos a analizar el rendimiento de los distintos algoritmos seleccionados.

Utilizamos Docker para montar cada una de estas herramientas y disponer de un entorno limpio y fácilmente replicable.

Para automatizar las pruebas y dotar de una interfaz gráfica, hemos usado HTML para la presentación, *Symfony* [16] (Framework de PHP) para realizar el sistema de automatización a la hora de lanzar las pruebas y medir el impacto de los algoritmos en función de en qué Frameworks se ejecute.

Estas herramientas nos permiten crear una solución potente y flexible a nuestras necesidades.

3.2 Análisis

3.2.1. Análisis de Requisitos

En este apartado vamos a realizar el análisis y elaboración de los requisitos asociados a este proyecto.

La construcción de los requisitos se basará en funcionales y NO funcionales. Para ello se elaborará una tabla en la que se recogen los distintos atributos de cada requisito.

Todos los requisitos (funcionales o no funcionales), se clasifican en distintas áreas temáticas en relación al tema que dictan, para así poder distinguirlos fácilmente.

A continuación, se muestra el formato de los requisitos:

IDENTIFICADOR	**TIPO/ÁREA**
DESCRIPCIÓN CORTA	
DESCRIPCIÓN LARGA	
PRUEBAS	

Tabla III Diseño de Tabla para los Requisitos

- **Área / Tipo:** atributo que contiene el área que ocupa el requisito de la aplicación.
- **Identificador:** atributo que da nombre al requisito mediante un código (TIPOREQ).
    - TIPOREQ puede adoptar dos valores: RF (**R**equisitos **F**uncionales) o RNF (**R**equisitos **N**o **F**uncionales).
- **Descripción corta:** atributo que contiene el propósito del requisito de manera muy breve.
- **Descripción larga:** atributo que explica de manera más extensa el requisito de la aplicación y expresa la funcionalidad.

Los requisitos los clasificamos según el área al que pertenecen, estas áreas están relacionadas con cada una de las fases de funcionamiento de la aplicación y son las siguientes:

- Generar prueba
- Ejecutar prueba
- Medir Rendimiento
- Almacenar Resultados

3.2.1.1. Requisitos Funcionales

Los requisitos funcionales, son los encargados de enumerar todas las características que tendrá nuestra aplicación, dichos requisitos son los siguientes:

IDENTIFICADOR	RF-001	TIPO/ÁREA	GENERAR PRUEBA
DESCRIPCIÓN CORTA	La aplicación permitirá crear pruebas desde la interfaz web		
DESCRIPCIÓN LARGA	La aplicación dispondrá de un formulario web desde el cual podremos lanzar pruebas para su ejecución y posterior análisis.		
PRUEBAS	Nos conectamos a la aplicación a través de un navegador y lanzamos una ejecución		

Tabla IV Requisito Funcional: RF-001

IDENTIFICADOR	RF-002	TIPO/ÁREA	GENERAR PRUEBA
DESCRIPCIÓN CORTA	La aplicación permitirá crear pruebas desde la línea de comandos		
DESCRIPCIÓN LARGA	La aplicación permitirá crear pruebas desde un Shell haciendo uso del comando curl se enviará la petición para su procesamiento y posterior análisis.		
PRUEBAS	Lanzamos una prueba a través de la Shell usando curl y comprobamos que se ha completado correctamente		

Tabla V Requisito Funcional: RF-002

IDENTIFICADOR	RF-003	TIPO/ÁREA	GENERAR PRUEBA
DESCRIPCIÓN CORTA	La aplicación almacenar todas las pruebas y sus resultados		
DESCRIPCIÓN LARGA	La aplicación almacenará todas las pruebas que se han ejecutado junto con sus resultados para poder analizarlos		
PRUEBAS	Ejecutamos dos pruebas y comprobamos que se ha almacenado toda la información.		

Tabla VI Requisito Funcional: RF-003

IDENTIFICADOR	**RF-004**	**TIPO/ÁREA**	**EJECUTAR PRUEBA**
DESCRIPCIÓN CORTA	La aplicación ejecutará las pruebas que estén en la cola		
DESCRIPCIÓN LARGA	La aplicación ejecutará las pruebas que se encuentren almacenadas y no se hayan ejecutado		
PRUEBAS	Lanzamos dos pruebas y comprobamos que todas terminaban		

Tabla VII Requisito Funcional: RF-004

IDENTIFICADOR	**RF-005**	**TIPO/ÁREA**	**EJECUTAR PRUEBA**
DESCRIPCIÓN CORTA	La aplicación ejecutará pruebas sobre Apache Flink		
DESCRIPCIÓN LARGA	La aplicación podrá ejecutar pruebas sobre el Framework Apache Flink		
PRUEBAS	Lanzamos una prueba sobre Apache Flink y comprobamos que termina		

Tabla VIII Requisito Funcional: RF-005

IDENTIFICADOR	**RF-006**	**TIPO/ÁREA**	**EJECUTAR PRUEBA**
DESCRIPCIÓN CORTA	La aplicación ejecutará pruebas sobre Apache Spark		
DESCRIPCIÓN LARGA	La aplicación podrá ejecutar pruebas sobre el Framework Apache Spark		
PRUEBAS	Lanzamos una prueba sobre Apache Spark y comprobamos que termina		

Tabla IX Requisito Funcional: RF-006

IDENTIFICADOR	**RF-007**	**TIPO/ÁREA**	**EJECUTAR PRUEBA**
DESCRIPCIÓN CORTA	La aplicación ejecutará pruebas sobre Apache Storm		
DESCRIPCIÓN LARGA	La aplicación podrá ejecutar pruebas sobre el Framework Apache Storm		
PRUEBAS	Lanzamos una prueba sobre Apache Storm y comprobamos que termina		

Tabla X Requisito Funcional: RF-007

IDENTIFICADOR	RF-008	TIPO/ÁREA	RESULTADOS
DESCRIPCIÓN CORTA	La aplicación medirá el rendimiento de las pruebas		
DESCRIPCIÓN LARGA	La aplicación medirá: • El Tiempo (Segundos) que tarda en completar una prueba. • El uso de CPU, teniendo en cuenta el número de procesadores / cores. • La cantidad de memoria RAM utilizada.		
PRUEBAS	Lanzamos una prueba y comprobamos las mediciones obtenidas son recibidas		

Tabla XI Requisito Funcional: RF-008

IDENTIFICADOR	RF-009	TIPO/ÁREA	RESULTADOS
DESCRIPCIÓN CORTA	La aplicación recibirá los resultados de las pruebas a través de RestFul		
DESCRIPCIÓN LARGA	Cuando una prueba termine, se enviarán los resultados desde el contenedor Docker vía curl		
PRUEBAS	Lanzamos una prueba y comprobamos las mediciones obtenidas son recibidas		

Tabla XII Requisito Funcional: RF-009

3.2.1.2. Requisitos NO Funcionales

Los requisitos no funcionales se encargan de describir todas las restricciones de nuestro sistema. Hemos identificados los siguientes requisitos que pertenecen a las siguientes categorías: **seguridad, rendimiento e interfaz**.

IDENTIFICADOR	**RNF-001**	**TIPO/ÁREA**	**RENDIMIENTO**
DESCRIPCIÓN CORTA	Solo puede existir una prueba en ejecución		
DESCRIPCIÓN LARGA	Para evitar la sobrecarga del sistema, este solo podrá ejecutar las pruebas de una en una. Hasta que no termine la ejecución de una prueba no se podrá lanza la siguiente		
PRUEBAS	Crear dos pruebas y comprobar que NO se ejecutan en paralelo		

Tabla XIII Requisito NO Funcional: RNF-001

IDENTIFICADOR	**RNF-002**	**TIPO/ÁREA**	**RENDIMIENTO**
DESCRIPCIÓN CORTA	Monitor de rendimiento		
DESCRIPCIÓN LARGA	La medición de los recursos consumidos por el sistema se limitará a una medición por segundo		
PRUEBAS	Comprobamos que las mediciones esta espaciadas un segundo		

Tabla XIV Requisito NO Funcional: RNF-002

IDENTIFICADOR	**RNF-003**	**TIPO/ÁREA**	**RENDIMIENTO**
DESCRIPCIÓN CORTA	Cola de pruebas		
DESCRIPCIÓN LARGA	La aplicación permitirá que se creen más de una prueba, las cuales se almacenarán y se irán procesando una por una		
PRUEBAS	Creamos dos pruebas y vemos que una de ellas se almacena		

Tabla XV Requisito NO Funcional: RNF-003

IDENTIFICADOR	RNF-004	TIPO/ÁREA	INTERFAZ
DESCRIPCIÓN CORTA	La interfaz de la aplicación se adapta a distintos tamaños de pantalla		
DESCRIPCIÓN LARGA	La aplicación dispondrá de una interfaz que se adapte al tamaño de pantalla de los dispositivos que la visiten, permitiendo una correcta visualización en PC, Móvil y Tablet		
PRUEBAS	Accedemos a la aplicación con distintos dispositivos y comprobamos que la interfaz se adapta correctamente.		

Tabla XVI Requisito NO Funcional: RNF-004

IDENTIFICADOR	RNF-005	TIPO/ÁREA	INTERFAZ
DESCRIPCIÓN CORTA	La aplicación permitirá graficar los resultados		
DESCRIPCIÓN LARGA	La aplicación graficará los resultados de las mediciones, siendo CPU y RAM consumidas y tiempo en completar el algoritmo		
PRUEBAS	Descargamos los datos de una prueba realizada		

Tabla XVII Requisito NO Funcional: RNF-005

IDENTIFICADOR	RNF-006	TIPO/ÁREA	RENDIMIENTO
DESCRIPCIÓN CORTA	La aplicación permitirá descargar las mediciones de rendimiento		
DESCRIPCIÓN LARGA	La aplicación dispondrá de un botón donde descargarnos los datos de las mediciones en formato CSV		
PRUEBAS	Descargamos los datos de una prueba realizada		

Tabla XVIII Requisito NO Funcional: RNF-006

3.3 Diseño

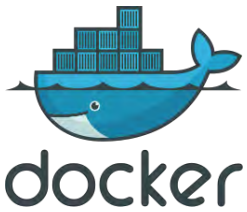
3.3.1. Elección de las Herramientas Utilizadas

En este apartado vamos a realizar una comparación entre las distintas herramientas seleccionadas y sus posibles alternativas.

3.3.1.1. Micro servicios

- Docker es un proyecto que nos permite crear entornos virtuales donde poder instalar software, realizar pruebas … sin que afecte al sistema donde se está ejecutando, una ventaja importante de Docker es que minimiza la sobrecarga que supone virtualizar todo un sistema operativo pues Docker hace uso de las características del kernel de Linux para lograr esta tarea.
- La alternativa que se planteó era usar Vagrant, el cual permite administrar distintas máquinas virtuales a través de un archivo de configuración haciendo uso de VMware / Virtualbox esta solución generaría una máquina virtual completa lo que nos garantiza todo un sistema operativo completo.

Para la tarea que queremos realizar consideramos que Docker es más apropiado por minimizar la sobrecarga y garantizar que cada inicio es la primera vez que se utiliza.
A continuación, se muestra una tabla comparativa con las principales características, necesarias en nuestro proyecto.



Sobrecarga	Mínima	Media
Multiplataforma	Sí	Sí
Mantiene los cambios en el S.O. Virtual	No	Sí
Archivo de Configuración	Si	Sí

Tabla XIX Comparativa Docker vs Vagrant

3.3.1.2. Interfaz de Usuario

- HTML, CSS y JS es el estándar que se está utilizando para la distribución de muchas aplicaciones, pues el cliente (Navegador WEB) necesario para su uso está extendido e implantado en empresas y hogar. La principal ventaja que tenemos, es que no hay que hacer ningún despliegue de cliente y que las modificaciones de la interfaz son instantáneas.

- Qt Jambi es una librería Open Source para java y c++ para el diseño de GUI (Interfaz Gráfica de Usuario) multiplataforma (Linux, OSX, Windows). Permite integrarse dentro del IDE eclipse, lo que facilita su utilización.

Finalmente hemos decidido utilizar el "Stack" HTML5+CSS3+JS como interfaz de usuario por su versatilidad y su amplio uso en el día a día de los usuarios.

		
Multiplataforma	Sí	Sí
Necesita mantener un cliente	No	Sí
Es un "Estándar"	Sí	No
Amplia documentación	Sí	No
modificaciones de interfaz inmediatas	Sí	No
Diseño Sensible al tamaño de pantalla	Sí	No

Tabla XX Comparativa HTML+CSS+JS vs Qt Jambi

3.3.1.3. Back – End

- Symfony3 es un Frameworks PHP que permite desarrollar aplicaciones web de forma rápida, dispone de ORM para acceso a la BBDD, gestión de plantillas, flexible (está desarrollado en PHP) y dispone de una alta comunidad. Este Frameworks nos ofrece un entorno muy completo para desarrollar aplicaciones web complejas de forma muy sencilla.

- ASP.NET es un Frameworks para aplicaciones web desarrollado por Microsoft y que ofrece toda la pila de componentes necesarios para el desarrollo de aplicaciones web. ASP.net se ejecuta sobre .NET, permitiendo que portabilidad del código entre plataformas.

Después de analizar los dos Frameworks elegimos Symfony, por su fácil y rápida configuración, su multiplataforma es mucho más madura, pues .net
se puede ejecutar fuera de Windows desde 2015.

	Symfony	ASP.net
Multiplataforma	Sí	Sí
Simplicidad	Sí	No
Configuración Rápida	Sí	No
Amplia documentación	Sí	Sí
Licencia	MIT	EULA

Tabla XXI Comparativa Symfony vs ASP.net

3.3.2. Diseño Final Elegido

Como primer punto de aproximación al diseño final, en el siguiente esquema se puede ver cómo interactúan entre ellos los distintos componentes.

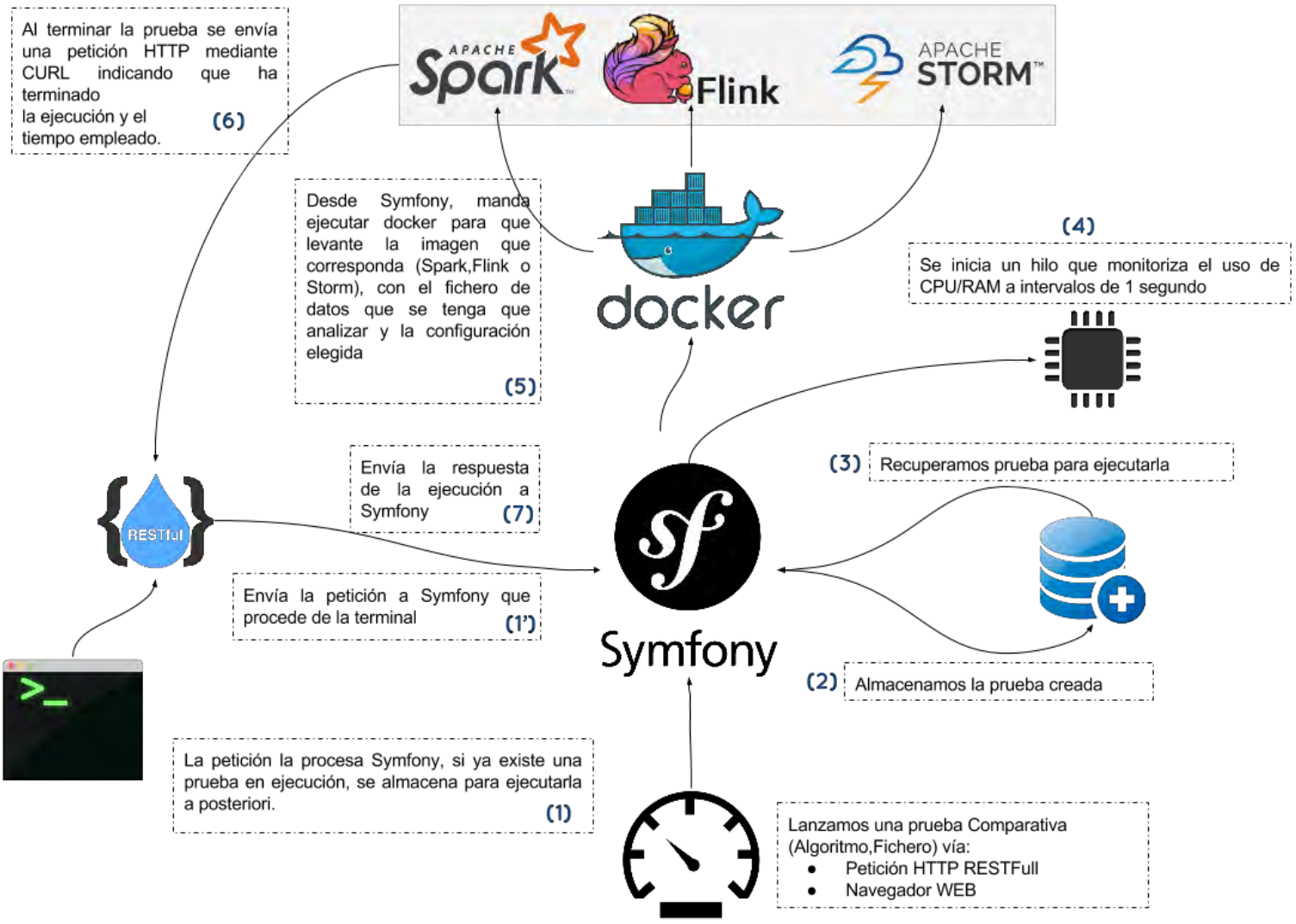


Ilustración XI Vista de la Arquitectura del Benchmark

A continuación, mostramos el Modelo Entidad-Relación utilizado para almacenar las pruebas, tanto las que ya han sido procesadas como las que quedan a la espera, la configuración que tiene cada prueba, su estado y el resultado de la misma.

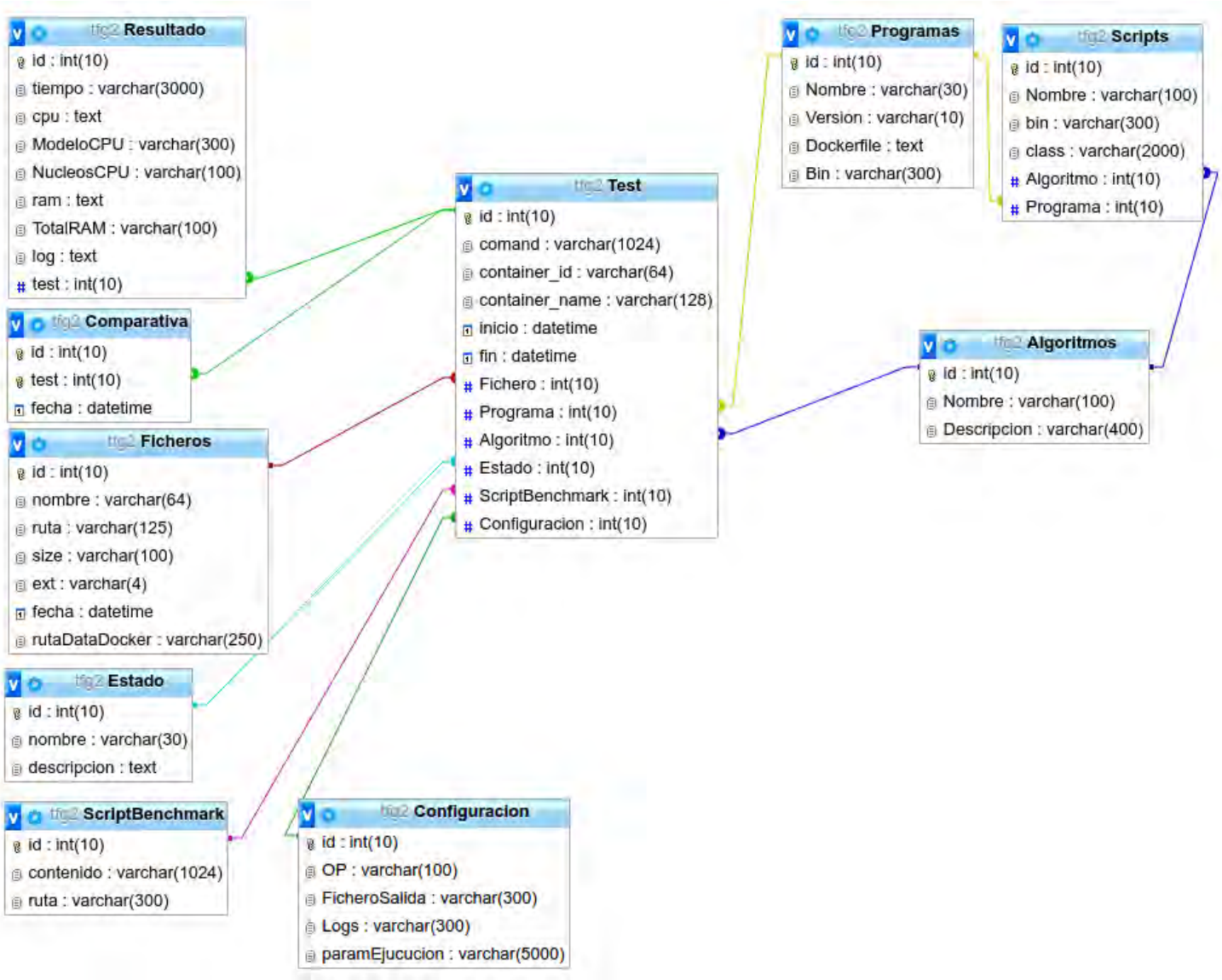


Ilustración XII Modelo Entidad-Relación del Benchmark

3.4 Implementación

En este apartado vamos a proceder a explicar cómo hemos realizado la implementación de cada componente de la arquitectura de la aplicación.

3.4.1. Crear Prueba

En este apartado debemos discernir dos formas de crear una prueba:

I. Individual: Esta prueba lo que permite es ejecutar uno de los tres Frameworks analizados con un fichero de datos concreto.
II. Comparativa: Este modo lo que permite es ejecutar los tres Frameworks con un mismo algoritmo y un archivo de datos, con el fin de comparar cual es más eficiente.

El primer caso solo genera una prueba mientras que el segundo tres.

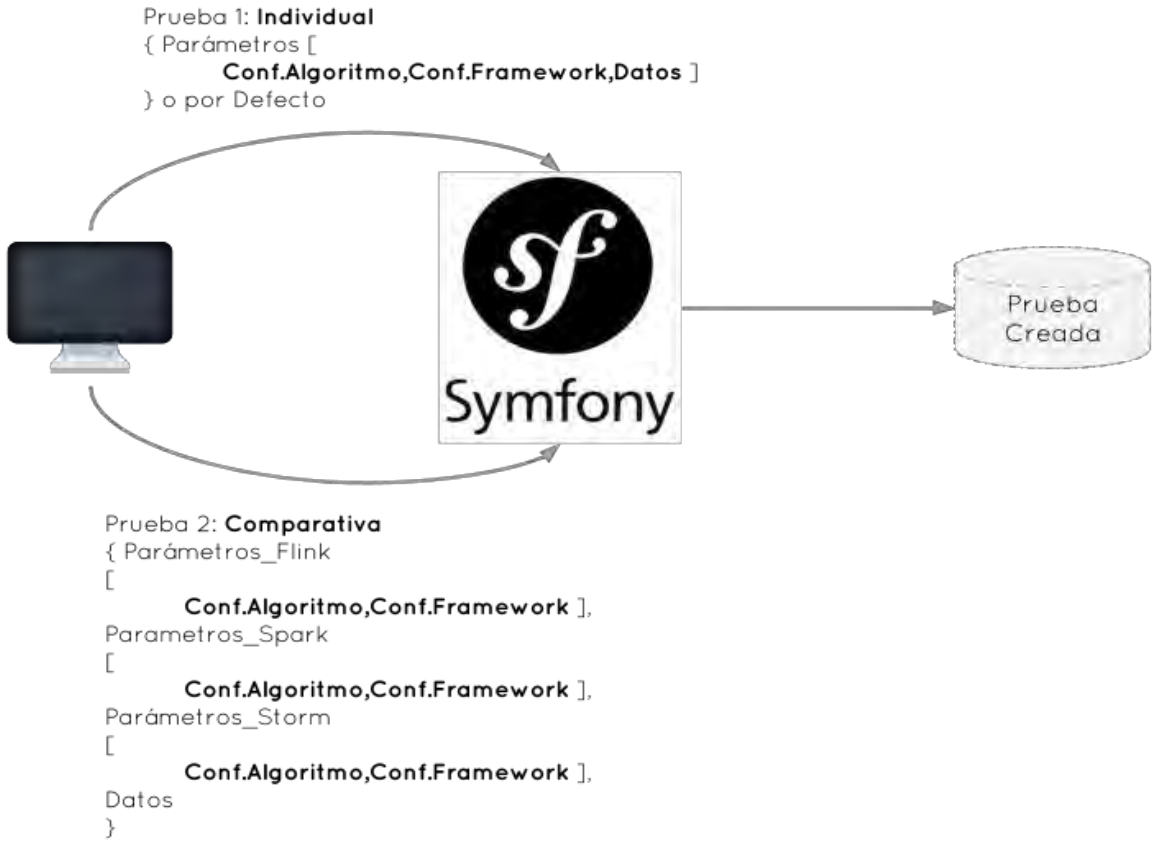


Ilustración XIII Explicación de cómo Crear Pruebas

3.4.2. Gestión de la Prueba

Cuando Symfony recibe una petición de prueba, realiza el siguiente proceso:

I. Recoge la configuración con la que se quiere ejecutar la prueba.
II. Genera una prueba, la cual consta de las siguientes partes:
    a. Se crea un fichero llamado medir_ {**FLINK, STOM, SPARK**}.sh, este fichero dispone de los elementos para ejecutar la prueba como son:
        i. Ruta hacia el Frameworks.
        ii. Algoritmo a ejecutar.
        iii. Entrada de datos.
        iv. Salida de datos.
    b. Se añade los comandos necesarios para medir el tiempo de ejecución.
    c. Se añade una llamada a Curl para enviar los datos desde Docker al servidor de Benchmark.
III. Si existe una prueba que se está ejecutando, toda esta información se almacena en la BBDD, creando una prueba con estado pendiente y que se ejecutará cuando le llegue su turno, el cual sigue una política **FIFO.**

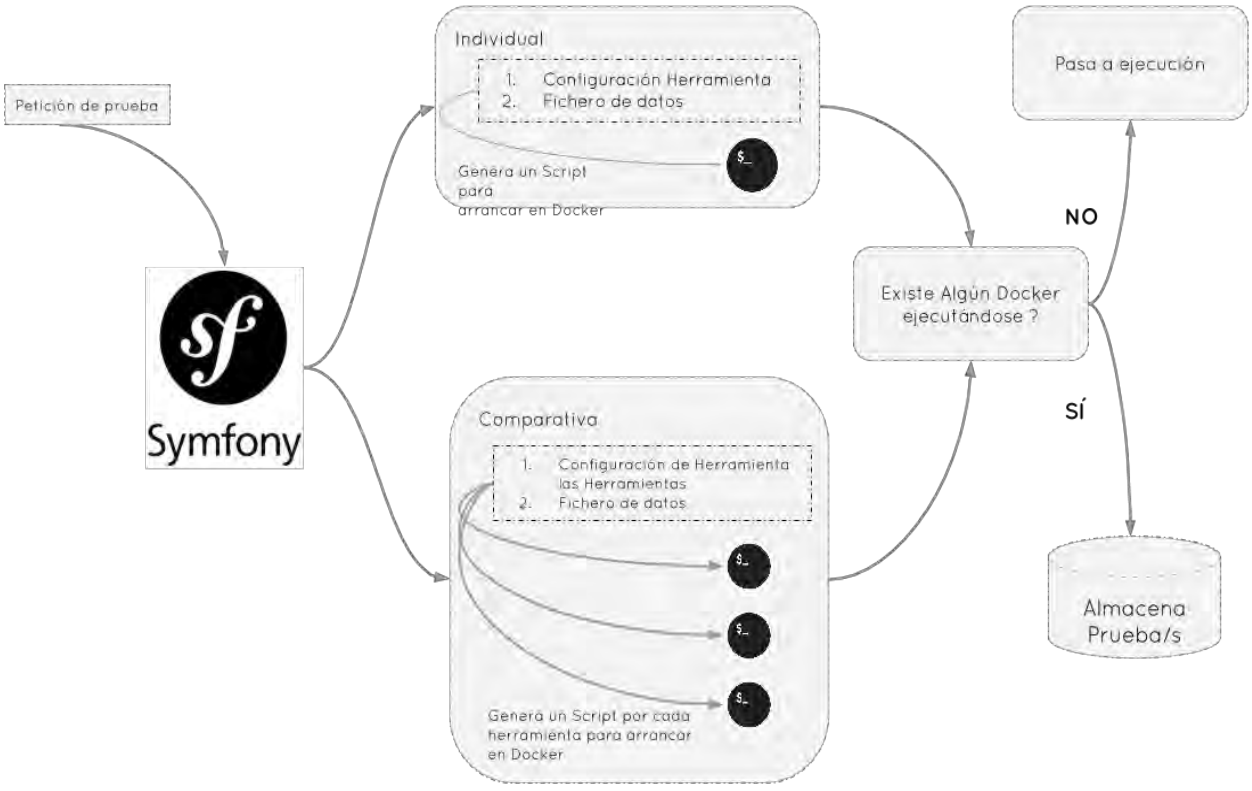


Ilustración XIV Explicación de cómo se gestionan las Pruebas

En el instante que una prueba pasa a ejecutarse, suceden las siguientes acciones:

I. Se comprueba que el servidor de monitorización del equipo está listo para medir el rendimiento del algoritmo, si está apagado se enciende.

II. Se obtiene de la BBDD el script y se escribe en un directorio asignado donde Docker puede acceder.

III. Symfony levanta un Docker con el siguiente comando.

```
docker run
--name {FLINK, SPARK, STORM}_11_07_2016_46_e001567866ae10c845149e1bdd9fb7c4 -m 4880m
-v var/www/default/TFG/src/BenchmarkBundle/Resources/public/tfg/resources/:/bigData/
-dit ubuntu-{FLINK,SPARK,STORM}:latest /bigData/medir_{FLINK,SPARK,STORM}.sh
```

1. **docker:** Nombre del binario que permite ejecutar Docker.
2. **run:** Indica que procedemos a ejecutar un contenedor.
3. **--name:** Nos permite indicar un nombre de contenedor con el cual referirnos.
4. **-m:** memoria disponible para el contenedor
5. **-v:** nos permite compartir una carpeta local dentro del contenedor, en este ejemplo **.../public/tfg/resources/** se comparte como /bigData.
6. **-dit:** Obliga a la imagen que vamos a ejecutar a ejecutarse en segundo plano.
7. **ubuntu-spark:lastest:** Nombre de la imagen Docker que vamos a ejecutar.
8. **/bigData/medir_SPARK.sh:** Este será el Script que se ejecutará en el momento que el contenedor esté levantado. [17]

IV. Al terminar la ejecución se enviar una petición al Benchmark con el tiempo de ejecución. Esto provoca que se detenga la imagen y proceda a ejecutar la siguiente prueba encolada.

En el siguiente esquema describimos el proceso que se realiza para la ejecución de una prueba, desde que enviamos la petición de prueba, creamos el Script, levantamos el contenedor y recogemos las mediciones.

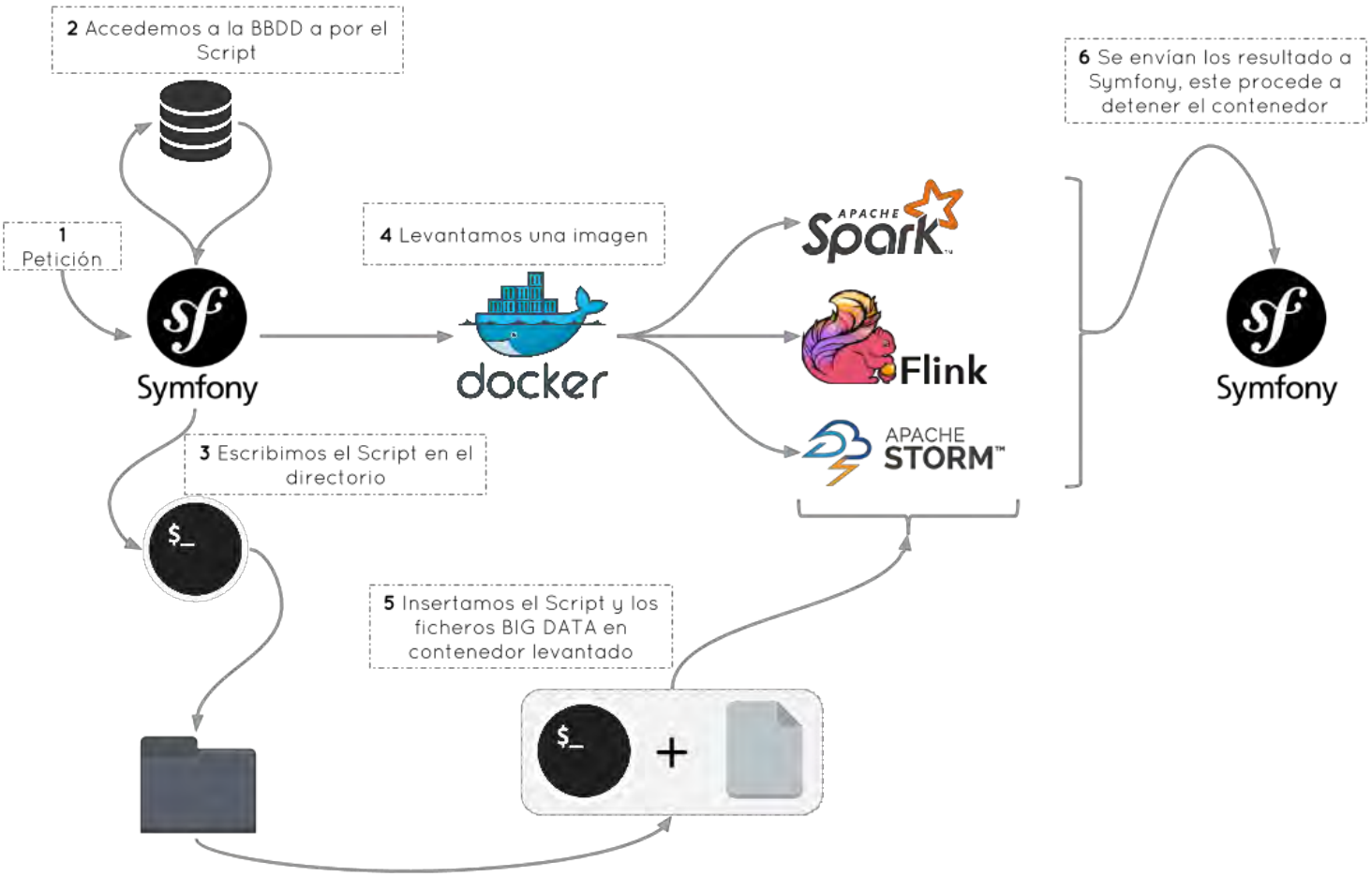


Ilustración XV Explicación de cómo se ejecutan las Pruebas

3.4.3. Compilación

Para poder ejecutar los algoritmos que permiten hacer uso de las características de los Frameworks es necesario compilar todos los binarios, en nuestro caso, vamos a realizar las pruebas sobre JAVA puesto que es el único lenguaje soportado por las tres herramientas de forma oficial.

El proceso cuenta de varias etapas:

I. Hacemos uso del comando **mvn eclipse:eclipse** el cual nos permite generar una estructura importable en eclipse.

II. Dentro de eclipse podemos ver cómo están estructuradas todas las librerías, modificar archivos y compilar. Para ello ejecutamos **clean package**, esto leerá un archivo XML con toda la estructura del proyecto y procederá a compilar todos los binarios y empaquetarlos en un fichero .JAR.

III. Una vez tenemos el JAR, ya podemos utilizar todos los binarios que hemos incluido por medio del XML. Para usar estos programas es necesario tener instalado todo el entorno de ejecución de los Frameworks.

En la siguiente imagen mostramos como se ve todas las librerías y ejemplos que trae Spark dentro de eclipse.

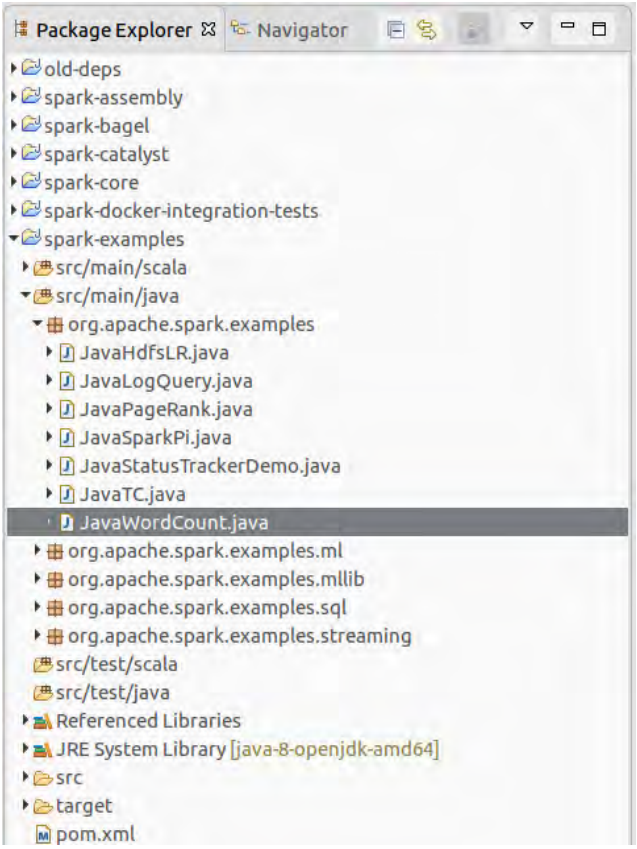


Ilustración XVI Librerías de Apache Spark

En esta otra imagen, podemos ver un fragmento del archivo XML que nos permite junto con maven empaquetarlo todo en un archivo JAR.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-parent_2.10</artifactId>
    <version>1.6.1</version>
    <relativePath>../pom.xml</relativePath>
  </parent>

  <groupId>org.apache.spark</groupId>
  <artifactId>spark-examples_2.10</artifactId>
  <properties>
    <sbt.project.name>examples</sbt.project.name>
  </properties>
  <packaging>jar</packaging>
  <name>Spark Project Examples</name>
  <url>http://spark.apache.org/</url>

  <dependencies>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_${scala.binary.version}</artifactId>
      <version>${project.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-streaming_${scala.binary.version}</artifactId>
      <version>${project.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-mllib_${scala.binary.version}</artifactId>
      <version>${project.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-bagel_${scala.binary.version}</artifactId>
      <version>${project.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
```

Ilustración XVII Vista del archivo XML de gestión de Proyectos Maven para Spark

3.4.4. Configuración de las Herramientas

Con el fin de mejorar el rendimiento de cada uno de los tres Frameworks vamos a ajustar a modificar la configuración para que se adapten mejor al hardware sobre el cual se van a ejecutar.

3.4.4.1. Apache Flink

Para cambiar la configuración es este Framework, debemos irnos a la ruta donde lo tengamos instalado, en nuestro caso:

/usr/local/src/Flink

Procederemos a ir a la carpeta **conf** y editaremos el archivo Flink-conf.yaml, donde las opciones más relevantes en nuestro caso son las siguientes:

- Jobmanager.heap.mg: Memoria que asignamos al jobManager para administrar los trabajos
- Taskmanager.heap.mg: Memoria que permitimos que use el taskManager para todas las tareas.
- Taskmanager.numberofTaskSlots: Esta variable nos permite indicar el número de huecos para la ejecución de tarareas. El valor más recomendable para esta opción es igual al número de núcleos físicos de nuestro/s procesador/es.
- Parallelism.default: Este valor debe ser igual o menor a *numberofTaskSlots.* Esto es debido a que la paralización se debe ejecutar en un hueco. [9]

```
35
36  # The heap size for the JobManager JVM
37
38  jobmanager.heap.mb: 512
39
40
41  # The heap size for the TaskManager JVM
42
43  taskmanager.heap.mb: 1024
44
45
46  # The number of task slots that each TaskManager offers. Each slot runs one parallel pipeline.
47
48  taskmanager.numberOfTaskSlots: 4
49
50
51  # The parallelism used for programs that did not specify and other parallelism.
52
53  parallelism.default: 4
54
55
```

Ilustración XVIII Archivo de Configuración de Apache Flink

3.4.4.2. Apache Spark

El caso de Spark disponemos de más opciones a la hora de configurar la herramienta:

I. La configuración se carga de forma dinámica en el momento que lanzamos la ejecutamos de un programa en Spark, predomina sobre la configuración global.

II. Podemos cambiar la configuración por defecto con la que se ejecutaran todos los programas. El archivo que debemos modificar es ***conf/spark-defaults.conf***.

Un ejemplo de la primera forma de ejecutar una aplicación sería la siguiente:

```
bin/spark-submit --master local[*] --driver-memory 3G --executor-memory 6G --class
org.apache.spark.tfg.JavaWordCount /bigData/script/spark/spark_2.10-1.6.1.jar datosIN
datosOUT
```

Donde:

- --local [*]: Con esta opción podemos indicar el número de núcleos/procesadores reales de CPU a utilizar para la ejecución del programa. Con asterisco, indicamos que utilice todos los disponibles de esta forma si lanzamos la ejecución en una plataforma con más núcleos no tendremos que modificar la configuración. Si queremos indicar un número concreto de núcleos a utilizar lo indicaríamos en el lugar que se encuentra el asterisco. [12]
- --driver-memory: Con esta opción configuramos la memoria que dispondrán los Workers.
- --executor-memory: Con esta opción configuramos la memoria que dispondrán los Executors.
- --Class: Indica que programa vamos a ejecutar.

La segunda forma de modificar la configuración, es cambiar el fichero spark-defaults.conf. Para ello debemos irnos a la ruta donde lo tengamos instalado Spark, en nuestro caso:

/usr/local/src/Spark

Si editamos el archivo, nos encontraos con un fichero de configuración **Clave, Valor**.

```
spark.master                     spark://master:7077
# spark.eventLog.enabled           true
# spark.eventLog.dir               hdfs://namenode:8021/directory
# spark.serializer                 org.apache.spark.serializer.KryoSerializer
spark.driver.memory              4g
spark.executor.memory              4g
```

Ilustración XIX Archivo de Configuración de Apache Spark

3.4.4.3. Apache Storm

Storm dispone de un archivo de configuración, el cual podemos editar parametrizar distintas opciones. Para modificar la configuración por defecto, nos dirigimos a la ruta de instalación de la herramienta, en nuestro caso:

/usr/local/src/Storm

Seguidamente, accedemos a la carpeta **conf** y abrimos el archivo **defaults.yaml.** Este fichero sigue una estructura de **clave:valor** y algunas de las opciones que podemos configurar son:

```
java.library.path: "/usr/local/lib:/opt/local/lib:/usr/lib"

### storm.* configs are general configurations
# the local dir is where jars are kept
storm.local.dir: "storm-local"
storm.zookeeper.servers:
    - "localhost"
storm.zookeeper.port: 2181
storm.zookeeper.root: "/storm"
storm.zookeeper.session.timeout: 20000
storm.zookeeper.connection.timeout: 15000
storm.zookeeper.retry.times: 5
```

Ilustración XX Archivo de Configuración de Apache Storm

- Ubicación de las librerías de Java.
- El puerto de escucha para el cordinador Zookeeper.
- Tiempo de espera para las sesiones.

Entre otras muchas opciones, las cuales podemos analizar desde la documentación oficial [18]

3.5 Implantación

Para realizar el despliegue de la aplicación clonamos el repositorio GIT https://github.com/MarioSkill/TFG-install.git usando el siguiente comando:

```
git clone https://github.com/MarioSkill/TFG-install.git
```

Una vez hemos obtenido el repositorio, procedemos a ejecutar como root el script. Durante la ejecución del mismo nos hará distintas preguntas sobre los componentes que se instalarán, así como de credenciales.

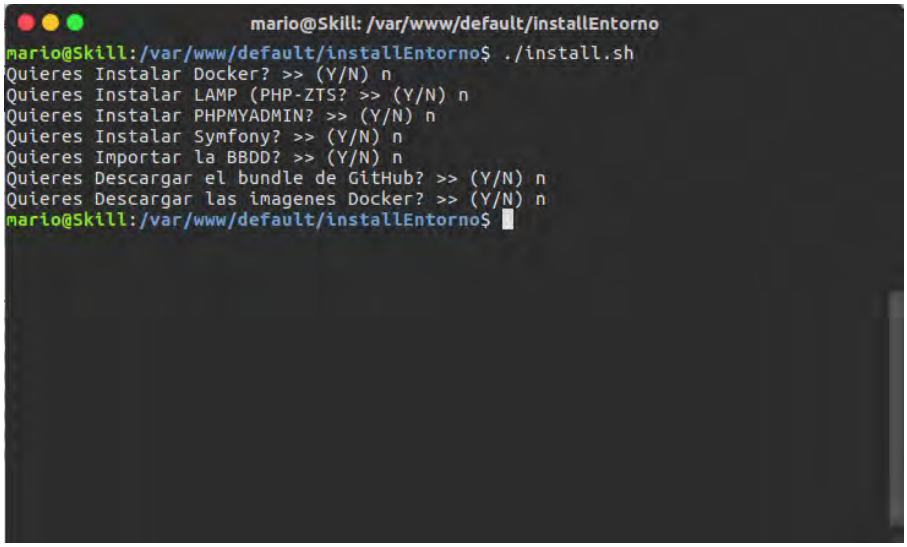


Ilustración XXI Como Instalar el Benchmark

Este procedimiento instalará los siguientes elementos:

- Docker
- Stack Lamp (con soporte multi-Hilo)
  - Apache 2
  - Mysql 5.6.28
  - PHP 5.6.18
- phpMyAdmin
- Symfony 3

Todos los componentes necesarios se encuentran disponibles en GitHub, para facilitar su distribución.

En el Anexo III, cuenta con una manual detallado de la instalación.

CAPÍTULO 4

4 Evaluación

En este apartado vamos a realizar un análisis del sistema diseñado, así como del soporte físico donde se va a ejecutar (características hardware).

4.1 Descripción del Entorno

El resultado de las pruebas que se van a ejecutar, están muy relacionadas con el hardware que las va a procesar, por este motivo es muy necesario conocer cuáles son las restricciones que vamos a tener que imponer, para que las pruebas se completen.

A continuación, indicamos las características hardware:

- Procesador[9]
  - Modelo: Intel Core i7 CPU 950
  - Frecuencia Base: 3.06 Ghz
  - Núcleos: 4
  - Hilos: 8
  - Memoria Caché: L1 y L2 privada de cada núcleo, L3 compartida
    - L1:
      - 4 x 32 KB de 4 vías asociativas para instrucciones
      - 4 x 32 KB de 8 vías asociativas para datos
    - L2: 4 x 256KB de 8 vías asociativas.
    - L3: 8MB de 16 vías asociativas.
- Memoria Principal:
  - Modelo:
  - Tamaño: 2048 MB x 3 Módulos
  - Frecuencia: 1066MHz

[9] Información obtenida de: http://www.cpu-world.com/CPUs/Core_i7/Intel-Core%20i7-950%20AT80601002112AA%20(BX80601950%20-%20BXC80601950).html

- Disco Duro[10]
  - Modelo: Samsung SSD 850 PRO
  - Tamaño: 256 GB
  - Lectura Secuencial: 550MB/s
  - Lectura Aleatoria Bloque 4KB: 100K IOPS*
  - Escritura Secuencial: 520 MB/s
  - Escritura Aleatoria Bloque 4KB: 90K IOPS*

El sistema operativo encargado de gestionar software y hardware es Ubuntu 16.04 LTS *Long Term Support* o soporte de largo plazo.

4.2 Resultados

4.2.1. Selección del conjunto de datos

A continuación, analizaremos el conjunto de datos seleccionado para cada algoritmo:

- WordCount:
  - Características Generales: Archivo de canciones Musicales, pueden tener 1 o más espacios de separación
  - Fichero1: 500 MB.
  - Fichero2: 1024 MB.
  - Fichero3: 2700 MB
- PageRank
  - Características Generales: Archivo formado de vértices y arista (Grafo).
  - Fichero1: Vértices: 20.000, Aristas: 20.000
  - Fichero2: Vértices: 40.000, Aristas: 40.000
  - Fichero3: Vértices: 80.000, Aristas: 80.000
- K-Menas
  - Características Generales: Fichero con puntos distribuidos en una dimensión 2 (plano x, y)
  - Fichero1: 112.000 Puntos
  - Fichero2: 448.000 Puntos
  - Fichero3: 896.000 Puntos

[10] Información obtenida de: http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/850pro.html

4.2.2. Selección de algoritmos

En este apartado explicaremos cuales han sido los algoritmos que evaluaremos y cómo funcionan.

4.2.2.1. Word Count

Este algoritmo como su propio nombre indica es contar palabras. Esta aproximación se la conoce como el "Hola Mundo" de los lenguajes de programación en las herramientas de análisis de datos.

Este algoritmo recibe un fichero o fuente de información del cual va obteniendo palabras, las cuales son contadas. A pesar de lo simple que parece esta implementación nos puede dar un buen indicativo de como de buena o mala es la herramienta, pues tiene que lidiar con grandes cantidades de información a la vez que la organiza y procesa. Este algoritmo encaja bastante bien con la filosofía de **MapReduce** [11]

4.2.2.2. PageRank

Este algoritmo de PageRank [19] PR en adelante, es uno de los métodos que Google utilizaba para determinar el orden de los resultados que se muestran en su buscador de internet. El artículo original "The Anatomy of a Large-Scale Hypertextual Web Search Engine" los podemos encontrar en Web http://infolab.stanford.edu/~backrub/google.html.

El algoritmo asigna un valor de PageRank a todas las páginas de internet, este valor es más alto si existen páginas que te enlazan.

La fórmula general que describe el algoritmo es el siguiente:

$$PR(A) = (1 - d) + d \sum_{i=1}^{n} \frac{PR(i)}{C(i)}$$

Donde:

- PR(A): Valor que obtiene la web.
- PR(i): Valor que tienen las webs que enlazan a "A" desde i hasta N.
- C(i): Numero de enlaces que salientes de la página i.
- d: Factor de amortiguación que reduce la influencia de páginas con mucha influencia, este valor se presupone que es de 0.85.
- (1-d): Valor que cuantifica los enlaces hacia sí misma.

NOTA: La fórmula de PageRank sigue una distribución de probabilidad, por lo que la suma de todas las webs no daría 1.

Con este algoritmo podremos comparar los diferentes modelos de gestión de memoria (estructuras utilizadas) y el particionamiento en subproblemas con el fin de acelerar el cálculo de la solución final.

4.2.2.3. K-Means

Este algoritmo de clustering [20], el cual particiona un grupo de n observaciones en K grupos donde cada observación n_i se asocia al grupo K_i, el cual tenga el valor medio más cercano.

$$J = \sum_{j=1}^{K} \sum_{i=1}^{n} \left\| X_i^{(j)} - C_j \right\| \text{^2}$$

Donde:

- K: Número de clusters.
- n: Número de casos.
- ‖x−c‖: Distancia euclidea.
- X_i: Caso i.
- C_j: Centroide del cluster j

En la siguiente imagen vemos cómo funciona el algoritmo:

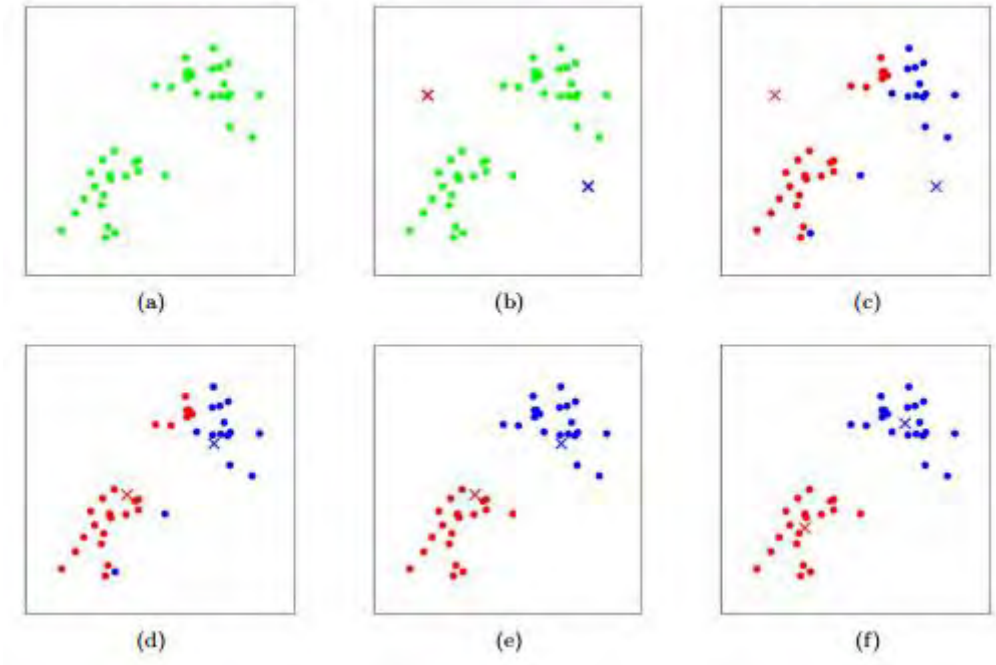


Ilustración XXII Funcionamiento de K-Means

Como se puede observar, partimos de un conjunto de datos, el cual queremos clasificar. El primer paso es elegir *K centroides* representados por X, normalmente estos centroides se pueden tomas de observaciones.
A continuación, empezamos a iterar y comparamos las observaciones restantes N-K, donde N es el número total de observaciones y K el número de centroides elegidos, y calculamos la distancia entre las observaciones restantes y los centroides seleccionados, asignado a cada observación el centroide K_i más cercano.
Una vez que todas las observaciones tienen asignado un centroide, lo cual genera K grupos o cluster, calculamos el nuevo centroide de cada grupo, y para ello calculamos la media de cada grupo y este será el valor del nuevo centroide.

Este proceso se repite hasta que los valores de las distancias apenas cambian.

4.2.3. Selección de métricas a medir

En este apartado analizaremos las métricas que hemos utilizado para medir el rendimiento que nos ofrece cada herramienta de análisis. Los indicadores que vamos a analizar son los siguientes:

- Tiempo que tarda el algoritmo en procesar los datos: Este indicador parece ser el más representativo pues el recurso más caro y el cual siempre vamos a querer minimizar.
- Uso y Evolución CPU: Este indicador nos medirá en cada momento cuales son los momentos de más carga computacional, en qué momento se producen y como se gestionan, de esta manera podremos graficar la evolución y ver una comparación rápida. **Este indicador obtiene la media de uso de CPU entre todos los núcleos.**
- Uso y Evolución RAM: Este tercer indicador nos informará del uso de memoria que está haciendo la herramienta de análisis y como las distintas estructuras de datos que utilizan ocupan más o menos memoria. Facilitará la comparación de la gestión de memoria que hace cada herramienta.

4.2.4. Configuración del entorno físico

En el apartado [4.1] de este documento, se puede obtener una visión general de la arquitectura física sobre la cual se han ejecutado la evaluación de los distintos algoritmos. Por limitaciones de **presupuesto**, estas pruebas han tenido que realizarse de forma local pero la naturaleza de arquitectura software permite fácilmente exportarla a un modelo de computación distribuida (está todo preparado, solo sería necesario indicar un listado de IP donde distribuir el cálculo).

4.2.5. Configuración de los proveedores de análisis

Vamos a realizar distintas configuraciones de las herramientas para evaluarlas, las restricciones impuestas sobre el problema, son las siguientes:

I. Restricción de Memoria:
   a. 1 GB.
   b. 2 GB.
   c. 4 GB.

II. Restricción de CPU.
   a. 1 Procesador.
   b. 2 Procesadores.
   c. 4 Procesadores.
   d. 8 Procesadores.

III. Restricción de Datos.
   a. Fichero 1: Menor tamaño
   b. Fichero 2: Tamaño intermedio.
   c. Fichero 3: Mayor tamaño.

En total generamos **36 configuraciones distintas** por cada algoritmo y herramienta de análisis que evaluemos.
Para saber cómo configurar las herramientas, nos podemos dirigir al capítulo [**3.4.4**] de este documento.

4.2.6. Ejecución de las pruebas

En este apartado graficaremos el resultado de las pruebas por cada algoritmo, con esto pretendemos mostrar de forma fácil y rápido todos los datos reunidos, con el fin de formar una conclusión sólida sustentada en los resultados obtenidos.

Las pruebas se van a realizar en las siguientes versiones de las herramientas de análisis:

- Apache Flink → 0.10.2
- Apache Spark → 1.6.1
- Apache Storm → 1.0.1
-

En el Anexo I: Resultados, podremos analizar los todos obtenidos de las pruebas y con los que hemos generado estos gráficos.

4.2.6.1. Evaluación Previa

En este apartado vamos a lanzar algunas ejecuciones para determinar cómo se comportan las herramientas con el fin de obtener un buen diseño de pruebas.

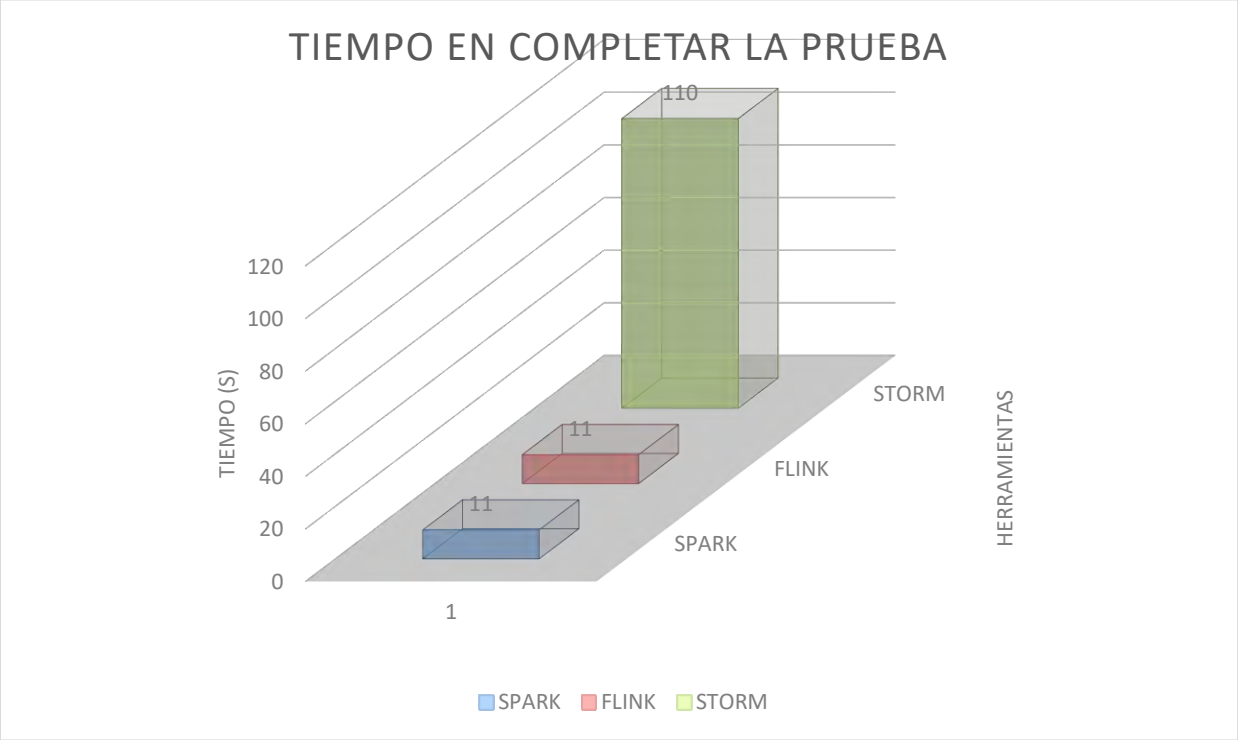
4.2.6.1.1. Evaluación I: 78 Megas

			
RAM PROMEDIO	50%	36.7%	42.38%
RAM MÁXIMO	56%	45%	
CPU PROMEDIO	63.52%	43.4%	40.24%
CPU MÁXIMO	87%	77%	
Tiempo(S)	**11**	**11**	**110**
Tamaño Datos	**78 Megas**		
Formato	**TXT**		
Resultado	**Terminado Correctamente**		

Tabla XXII Resultados Prueba I

En la siguiente imagen podemos ver como ha sido la evolución de uso de Memoria RAM de cada Framework durante la ejecución de la prueba.

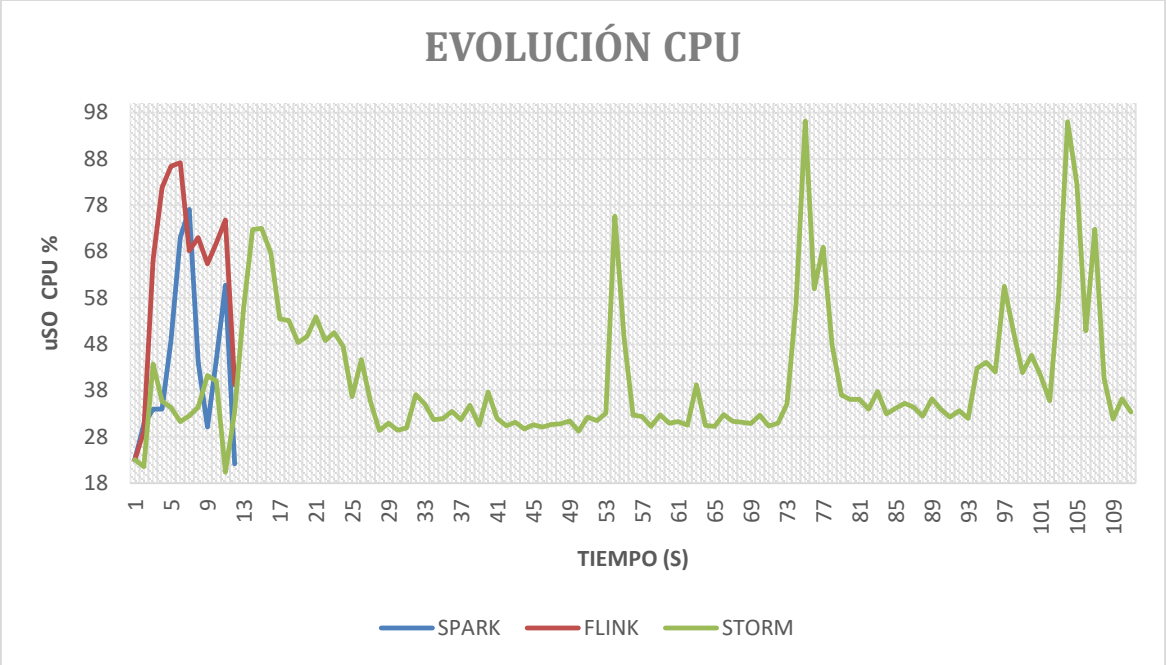
Vemos como Spark y Flink, completan el trabajo en 11 segundos, lo sorprendete es el tiempo (110 segundos) que necesita Storm en completar un trabajo que debería ser trivial por el tamaño del archivo.



Gráfica I Evaluación I: Comparación de Tiempo

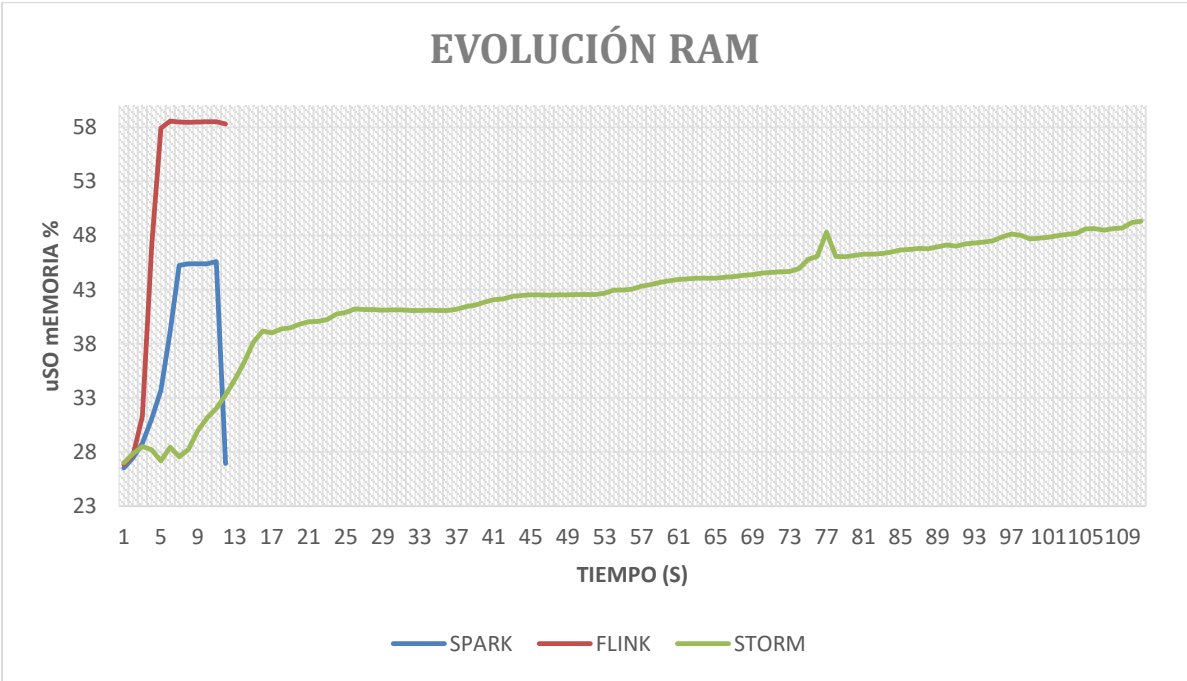
Si analizamos el uso de CPU se observa como Flink y Spark tienen un comportamiento similar, pero se puede apreciar una característica fundamental de cada herramienta. En Flink vemos como la gráfica evoluciona más constante mientas que Spark, vemos dos picos fuertes, esto se debe a que Spark procesa las tareas en bloques, mientras que Flink en flujo.

En Storm, los picos altos muestran cuando se están pasando entre las 2 estructuras que se utilizan (Spout y Bolt), el resto del tiempo también es bastante continuo, esto se debe a que utiliza la misma filosofía que Flink, el último pico se debe a la escritura en Disco.



Gráfica II Evaluación I: Evolución de la CPU utilizada SPARK vs FLINK vs STORM

En cuanto al uso de RAM, Flink y Spark, carga completamente los archivos en memoria, Storm por su parte va procesándolo poco a poco.



Gráfica III Evaluación I: Evolución de la RAM utilizada SPARK vs FLINK vs STORM

4.2.6.1.2. Evaluación II: 500 Megas

			
RAM PROMEDIO	57%	76%	65 %
RAM MÁXIMO	59%	85%	71%
CPU PROMEDIO	60%	76%	83%
CPU MÁXIMO	93%	99%	99%
Tiempo(S)	45	72	2520 = 42 minutos
Tamaño Datos	**500 Megas**		
Formato	**TXT**		
Resultado	**Terminado Correctamente**	**Error: Java heap space**	**Cancelado Manualmente**

Tabla XXIII Resultados Evaluación II

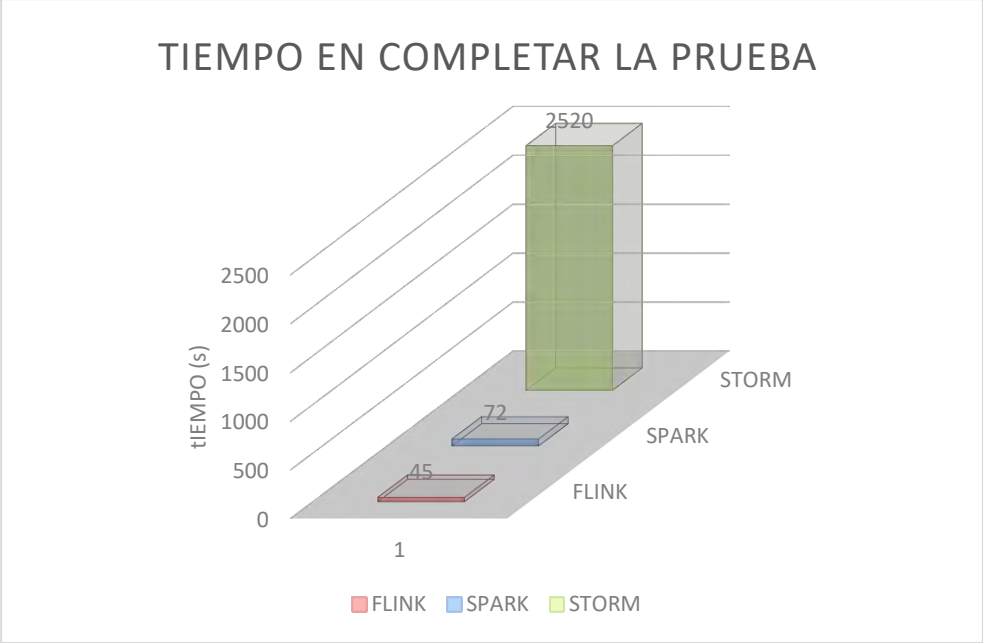
En el siguiente gráfico podemos ver como ha sido la evolución de uso de Memoria RAM de cada Framework durante la ejecución de la prueba.

Flink es el único en completar correctamente el procesamiento de los datos, realizando el proceso en **45 segundos**.

Spark no tiene suficiente memoria para completar la ejecución y falla cuando lleva **72 segundos**.

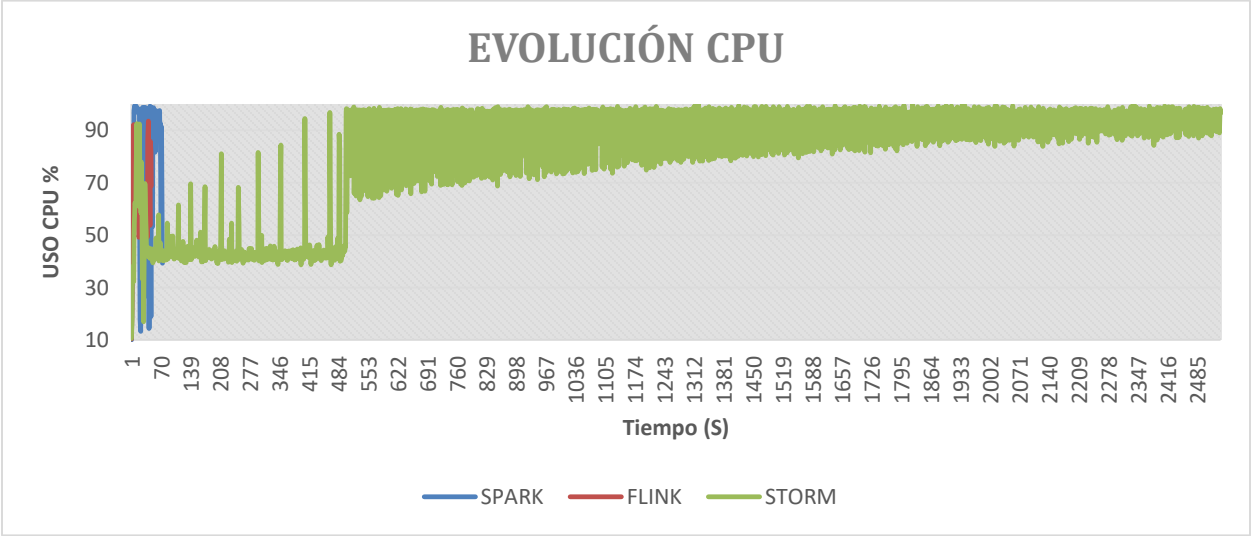
Storm necesita un tiempo totalmente desproporcionado para intentar procesar los datos, lo que se puede deber a que esta herramienta esta enfocada al procesamiento en Streaming y no en batch, en una estructura distribuida, con la cual no contamos.

En este primer gráfico se puede ver muy claramente la diferencia de tiempo que hay principalmente entre Flink, Spark y Storm.



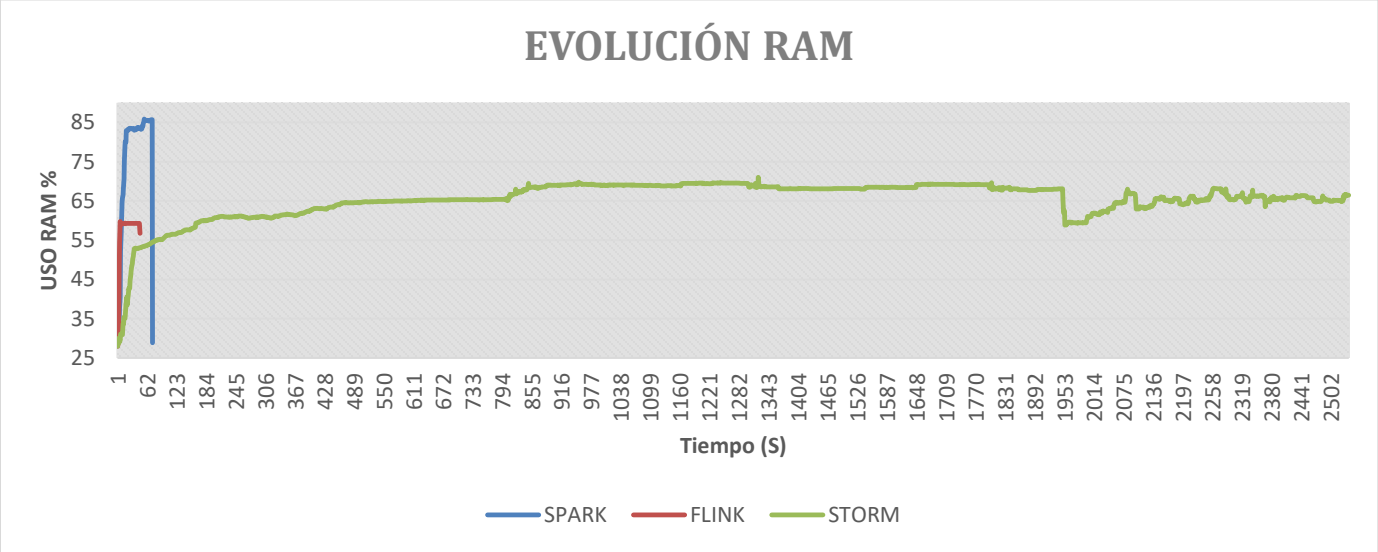
Gráfica IV Evaluación II: Comparación de Tiempo

En la evolución de CPU de los tres Frameworks, la conclusión que se observa es que Storm va aumentando el uso de todos los procesadores en los 42 minutos que se está ejecutando, pues el uso máximo sigue siendo 100% pero el mínimo al final de la prueba se sitúa en el 85% aproximadamente.



Gráfica V Evaluación II: Evolución de la CPU utilizada SPARK vs FLINK vs STORM

En la evolución de RAM, como hay un aumento de memoria en los primeros segundos de ejecución, esto se debe a que se realiza la carga del fichero a memoria para su procesamiento. Se observa una diferencia significativa en el uso que hace Spark con referencia a los otros dos. Esto se puede deber al tipo de estructura que utiliza para almacenar y procesar los datos. El uso que hace Storm, lo podemos considerar dentro de la normalidad pues se encuentra algo por encima del 65%.



Gráfica VI Evaluación II: Evolución de la RAM utilizada SPARK vs FLINK vs STORM

Después de realizar estas evaluaciones previas, concluimos que no disponemos de la infraestructura necesaria para poder ejecutar los algoritmos seleccionados en STORM, por lo que las **comparaciones finales se realizaran entre Apache FLINK y Apache SPARK.**

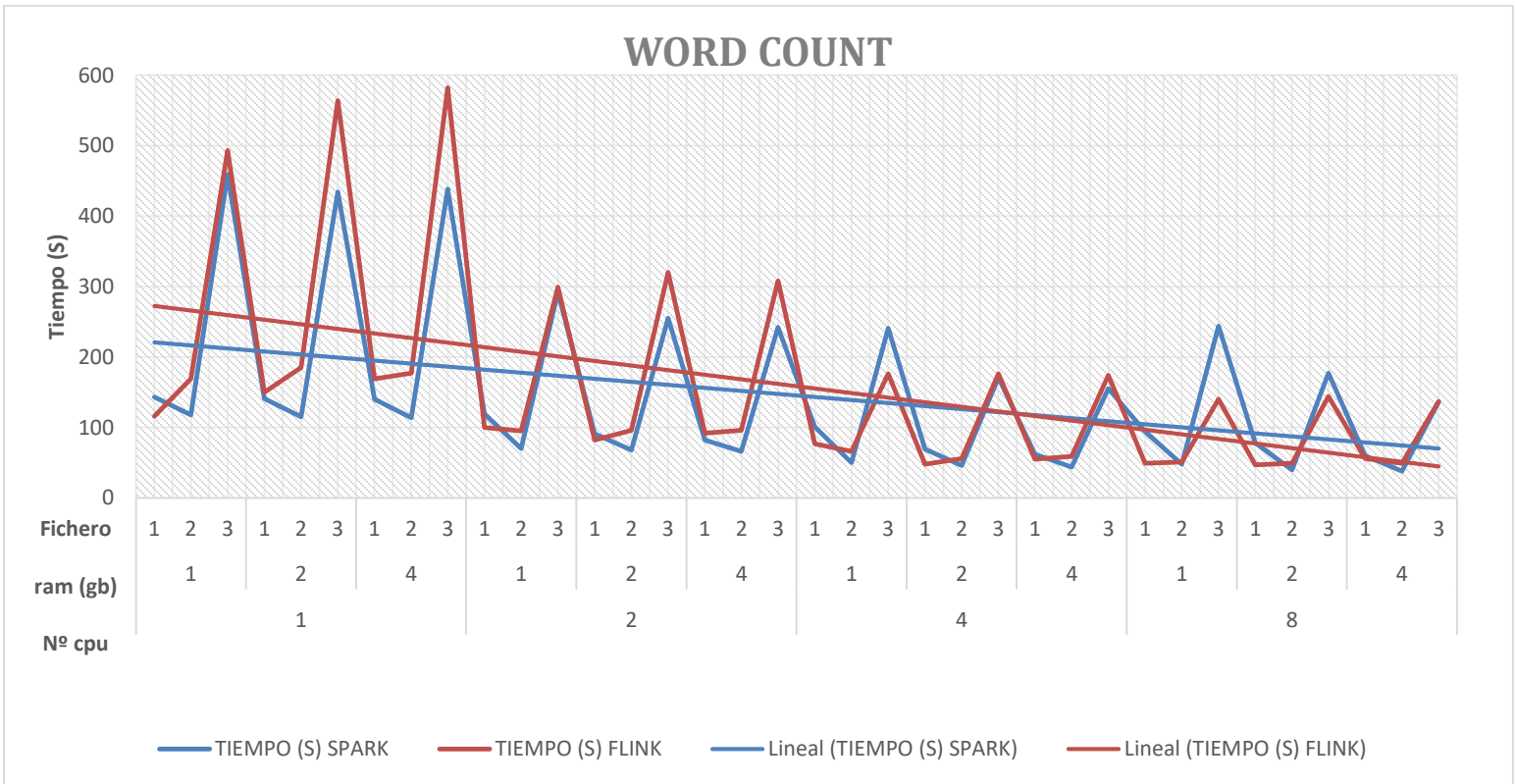
4.2.6.2. WordCount

En la siguiente tabla se especifica la configuración seleccionada para ejecutar Word Count en Spark y Flink.

CONFIGURACIÓN DE LA PRUEBA			
NOMBRE	TAMAÑO	MEMORIA (GB)	Nº PROCEADORES
FICHERO 1	500 MB	1,2 y 4	1,2,4 y 8
FICHERO 2	1024 MB	1,2 y 4	1,2,4 y 8
FICHERO 3	2700 MB	1,2 y 4	1,2,4 y 8
NÚMERO DE PRUEBAS UNICAS			36
REPETICIONES POR PRUEBA			3

Tabla XXIV WordCount: Configuración Prueba

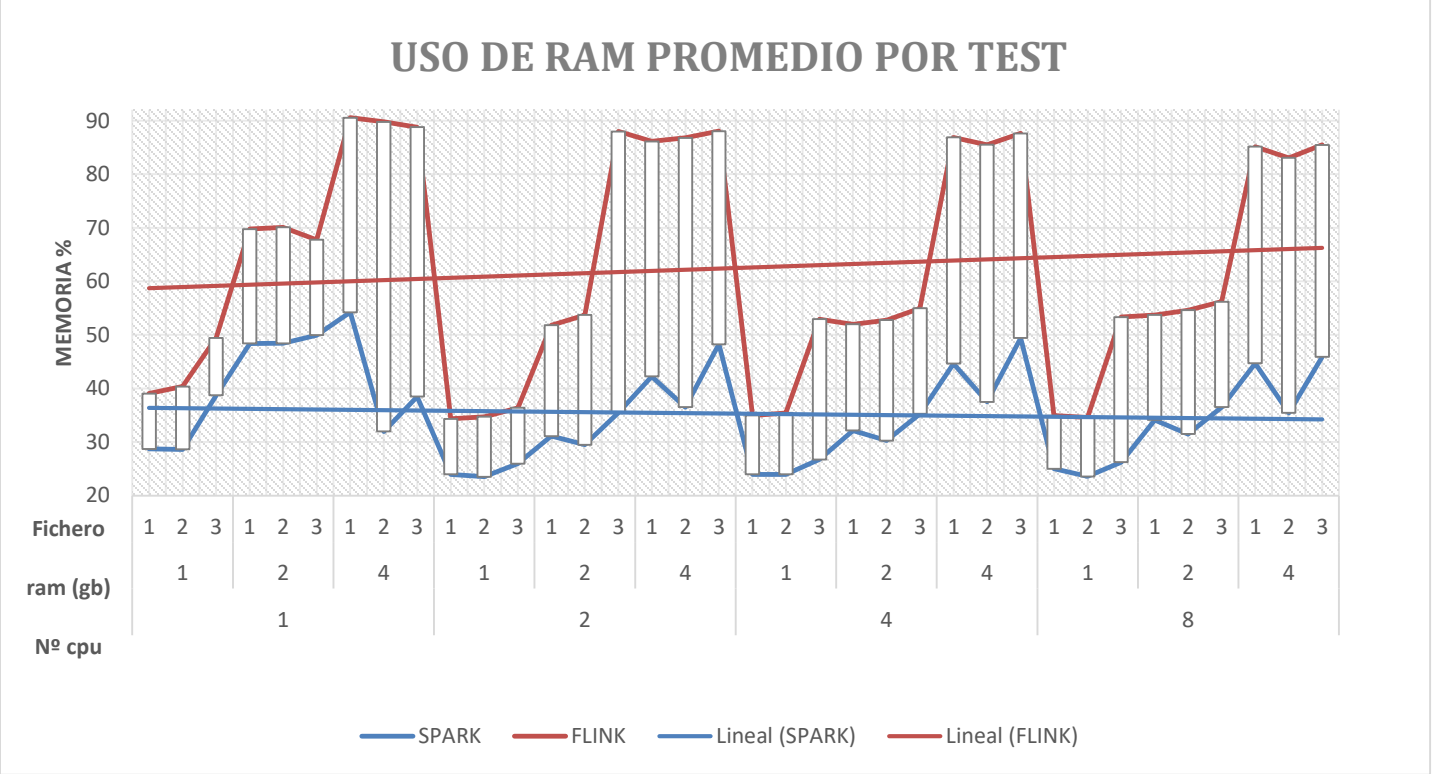
El gráfico muestra la cómo se comportan (tiempo medio de ejecución) ambas herramientas de análisis al procesar el algoritmo de contar palabras.



Gráfica VII WordCount: Spark vs Flink - Tiempo

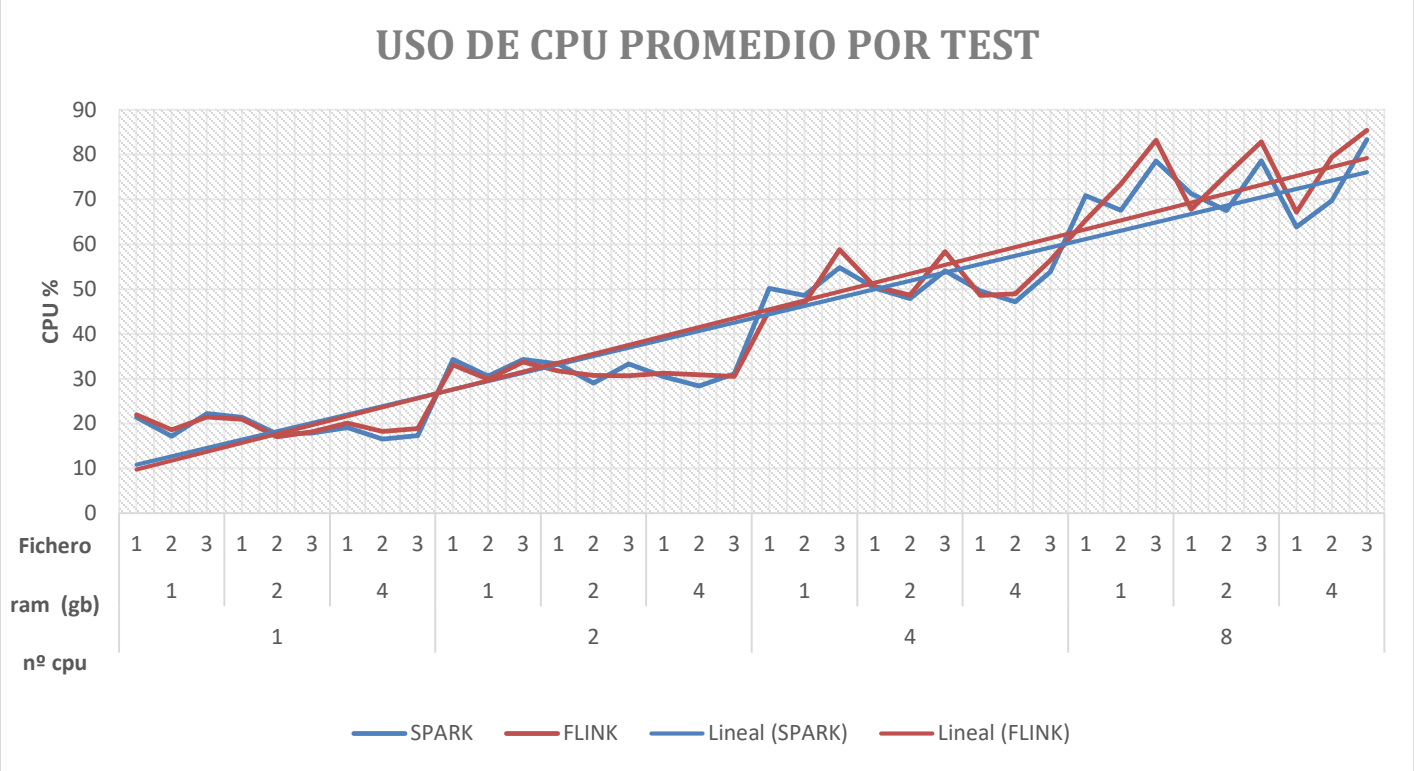
Se observa como Flink con uso procesador se comporta peor que Spark con una diferencia media de 55,8 Segundos más lento, pero al aumentar el número de procesadores y la memoria asignada, se observa como el tiempo en completar la prueba se reduce hasta que, con la configuración de 8 Procesadores, 1 GB de RAM los tiempos empiezan a favorecer a Flink.

El comportamiento en cuanto al uso de RAM, se observa Flink hace uso prácticamente de toda la memoria asignada por prueba, mientras que SPARK no necesita consumir toda la memoria.



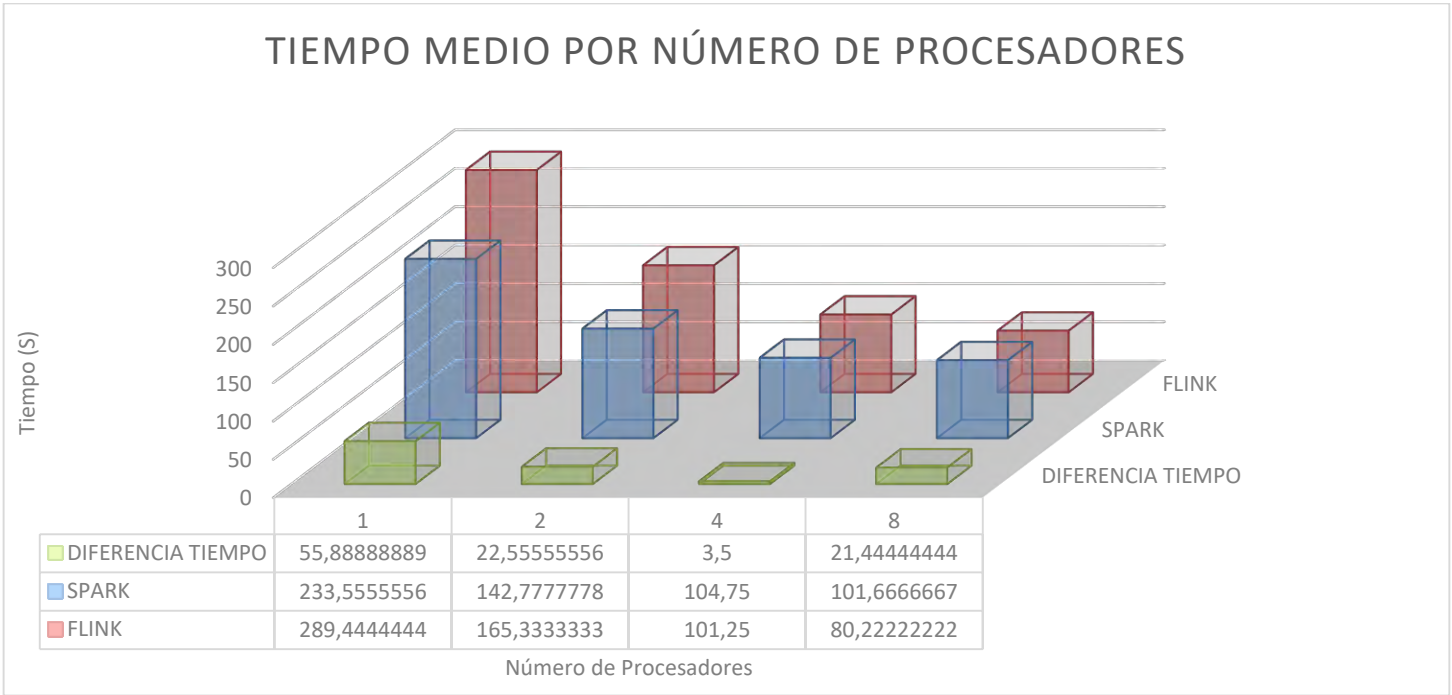
Gráfica VIII WordCount: Uso de Ram Promedio por Test

En la gráfica que vemos a continuación se observa el uso de CPU que consumen para procesar el fichero, observamos que las diferencias no son significativas hasta la ejecución con 8 procesadores, donde Flink está haciendo un uso más completo de ellos. En estas ejecuciones buscamos que las herramientas usen al máximo todos los recursos que les asignamos con el fin de que se reduzca el tiempo.



Gráfica IX WordCount: Uso de CPU promedio por Test

En esta otra gráfica se observa muy bien como al aumentar los recursos asignados a las herramientas, Flink consigue ir reducir el tiempo en completar las pruebas.



	1	2	4	8
DIFERENCIA TIEMPO	55,88888889	22,55555556	3,5	21,44444444
SPARK	233,5555556	142,7777778	104,75	101,6666667
FLINK	289,4444444	165,3333333	101,25	80,22222222

Gráfica X WordCount: Comparación de Tiempo por Nº de procesadores usados

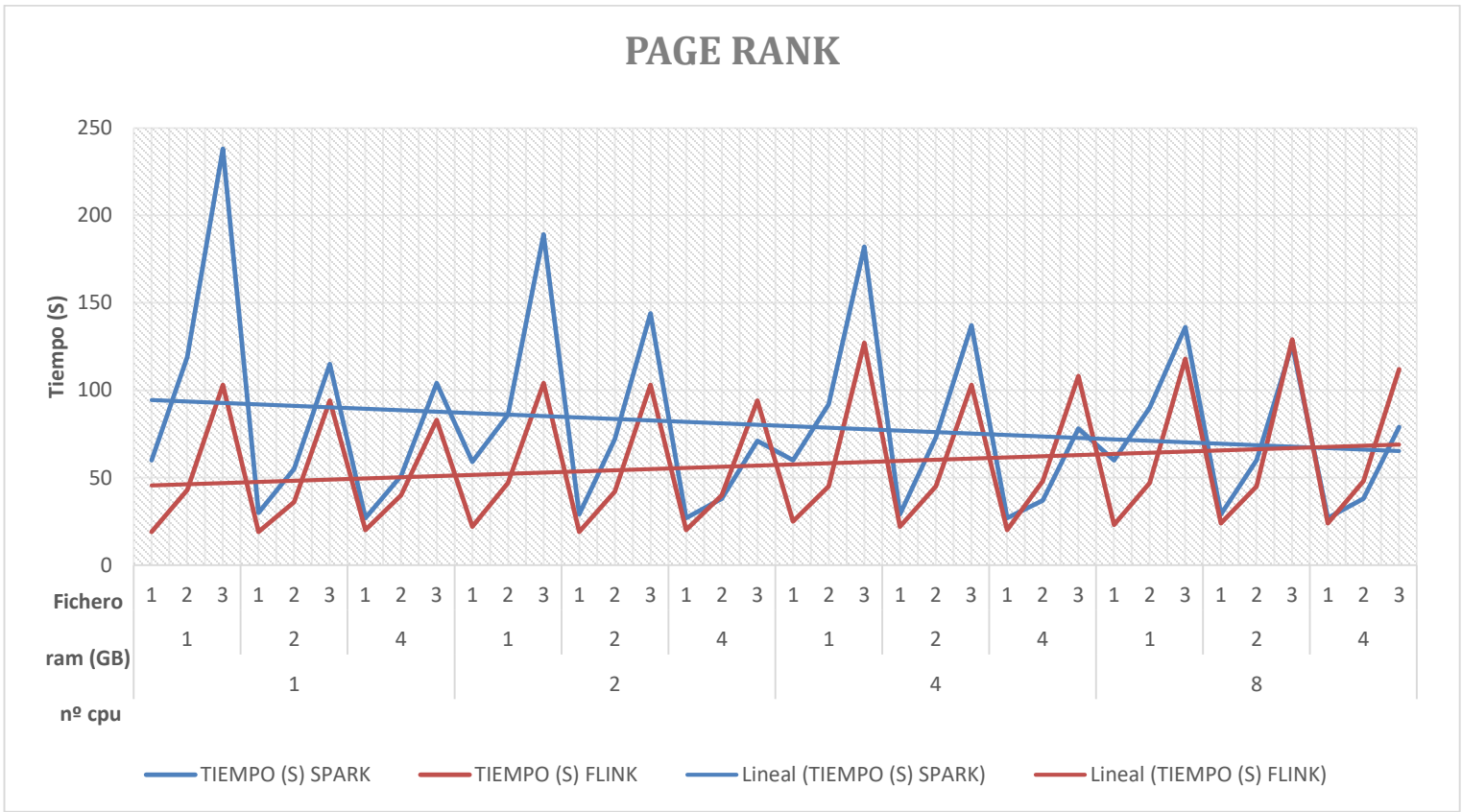
4.2.6.1. PageRank

En la siguiente tabla se especifica la configuración seleccionada para ejecutar Page Rank en Spark y Flink.

CONFIGURACIÓN DE LA PRUEBA			
NOMBRE	(V,A)	MEMORIA (GB)	Nº PROCEADORES
FICHERO 1	20.000, 20.000	1,2 y 4	1,2,4 y 8
FICHERO 2	40.000, 40.000	1,2 y 4	1,2,4 y 8
FICHERO 3	80.000, 80.000	1,2 y 4	1,2,4 y 8
NÚMERO DE PRUEBAS UNICAS			36
REPETICIONES POR PRUEBA			3

Tabla XXV PageRank: Configuración Prueba

El gráfico muestra la cómo se comportan (tiempo medio de ejecución) ambas herramientas de análisis al procesar el algoritmo de Page Rank.



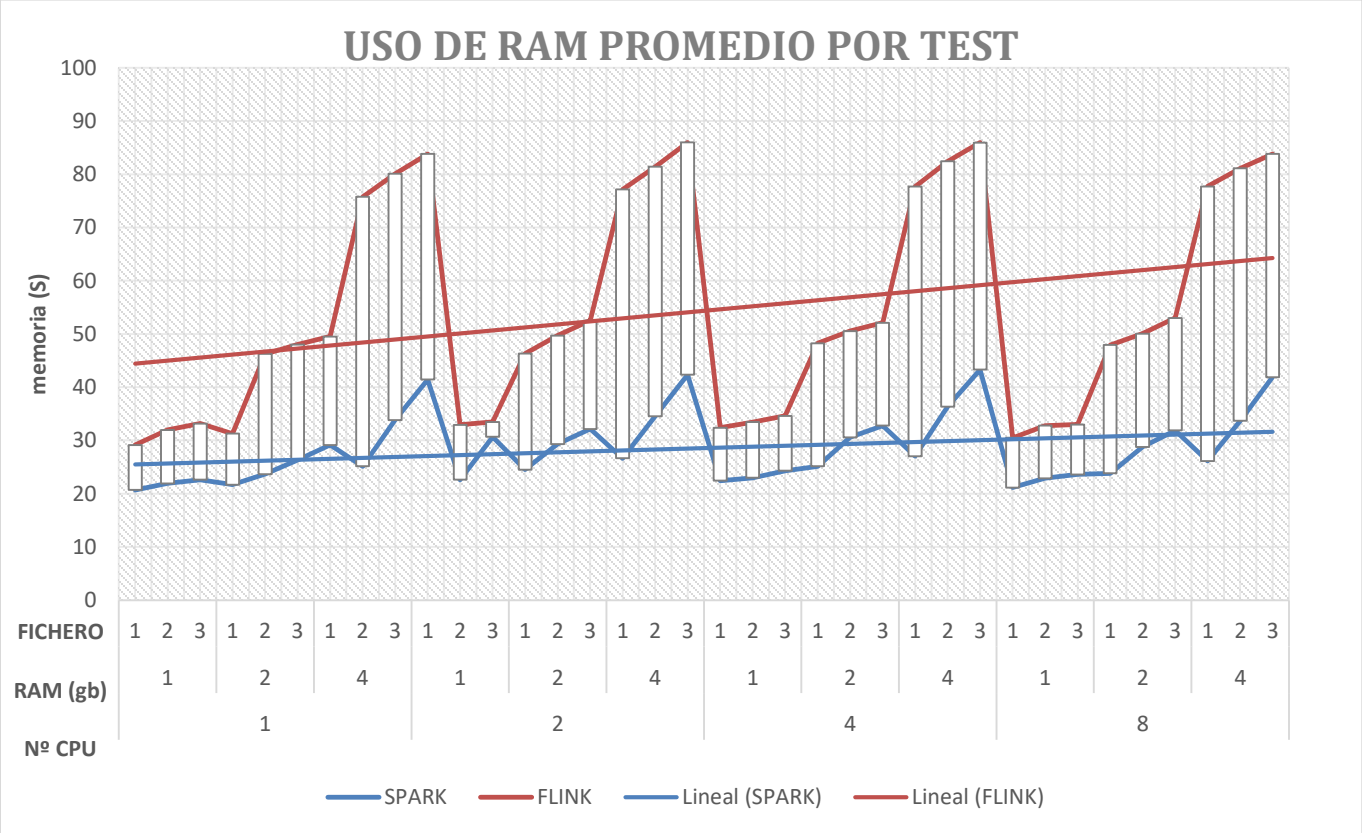
Gráfica XI PageRank: Saprk vs Flink – Tiempo

Analizando los resultados, observamos como Spark se comporta especialmente mal con un solo procesador y un Gigabyte de RAM, este es el test donde existe la mayor diferencia de rendimiento, pero esta se empieza a igualar al aumentar el uso de memoria y procesadores.

El caso de Flink es interesante, pues a pesar de que el rendimiento se va igualando, con una menor cantidad de memoria, obtiene mejor resultados independientemente del número de procesadores.

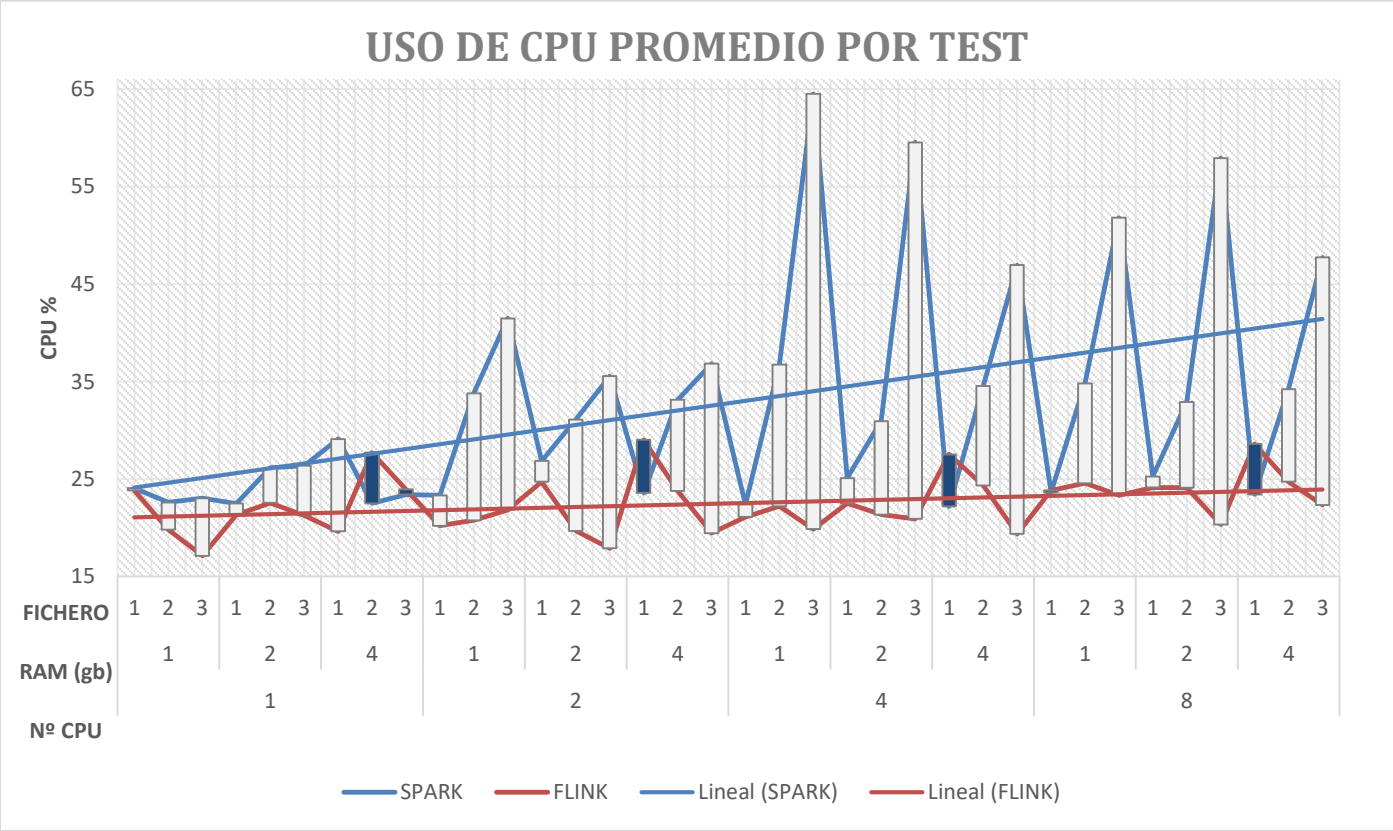
El comportamiento en cuanto al uso de RAM, se observa Flink hace uso prácticamente de toda la memoria asignada por prueba, mientras que SPARK no necesita consumir toda la memoria.

Lo que es bastante sorprendente es la poca memoria que necesita Spark para ejecutar la prueba.



Gráfica XII PageRank: Uso de RAM Promedio por Test

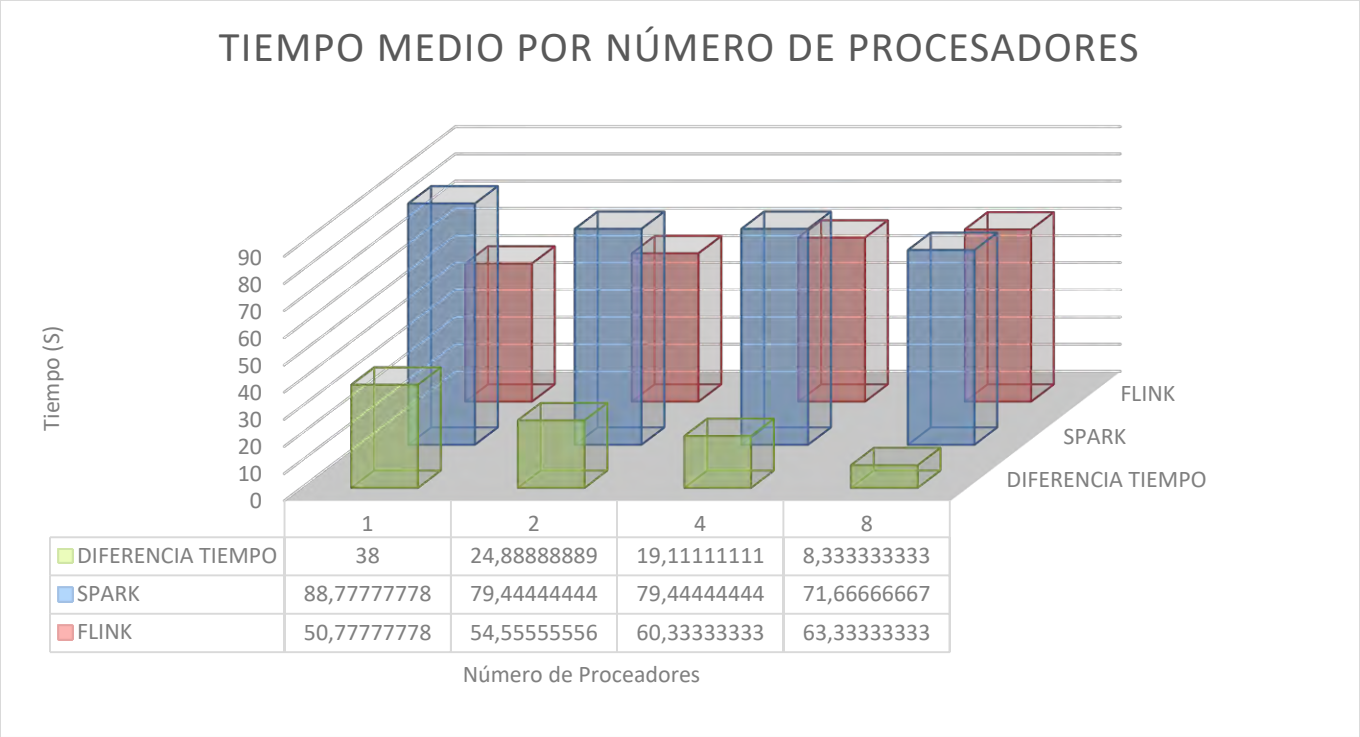
En la gráfica que vemos a continuación se observa el uso de CPU que consumen para procesar el fichero, observamos que existen grandes diferencias. Flink se mantiene constante entorno al 25% de uso de CPU mientras que Spark muestra un uso más irregular, esto lo podemos achacar al espíritu de Spark el cual está diseñado para ejecutar en Bach.



Gráfica XIII PageRank: Uso de CPU Promedio por Test

Por último, podemos ver cómo ha ido evolucionando el tiempo en completar las distintas pruebas ejecutadas y la diferencia de tiempo que existe entre Spark y Flink en función del número de procesadores que utilicemos.

En este caso, se observa como Spark reduce su tiempo de procesamiento en casi 20 segundos mientras que Flink, a pesar de seguir tardando menos, ha empeorado sus resultados con respecto a los realizados con menos recursos. Esta consecuencia la atribuimos a que la madurez de la librería de Grafos no sea la suficiente y no trabaje correctamente con la ejecución en paralelo.



	1	2	4	8
DIFERENCIA TIEMPO	38	24,88888889	19,11111111	8,333333333
SPARK	88,77777778	79,44444444	79,44444444	71,66666667
FLINK	50,77777778	54,55555556	60,33333333	63,33333333

Gráfica XIV PageRank: Comparación de Tiempo por Nº de procesadores usados

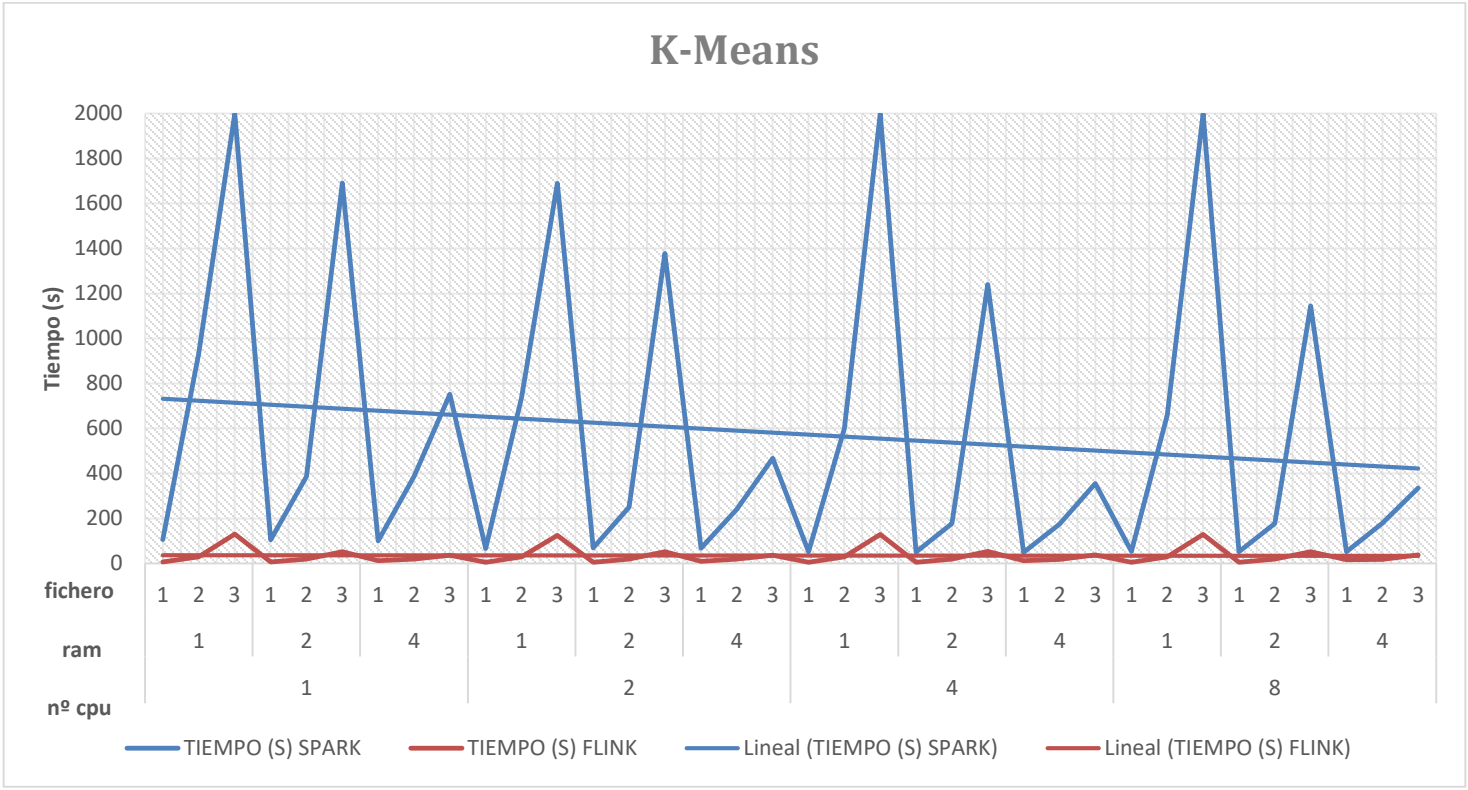
4.2.6.2. K-Means

En la siguiente tabla se especifica la configuración seleccionada para ejecutar K-Means en Spark y Flink.

CONFIGURACIÓN DE LA PRUEBA			
NOMBRE	(D,P)	MEMORIA (GB)	Nº PROCEADORES
FICHERO 1	2,112.000	1,2 y 4	1,2,4 y 8
FICHERO 2	2,448.000	1,2 y 4	1,2,4 y 8
FICHERO 3	2,896.000	1,2 y 4	1,2,4 y 8
NÚMERO DE PRUEBAS UNICAS			36
REPETICIONES POR PRUEBA			3

Tabla XXVI K-Means: Configuración Prueba

El gráfico muestra la cómo se comportan (tiempo medio de ejecución) ambas herramientas de análisis al procesar el algoritmo de K Medias.

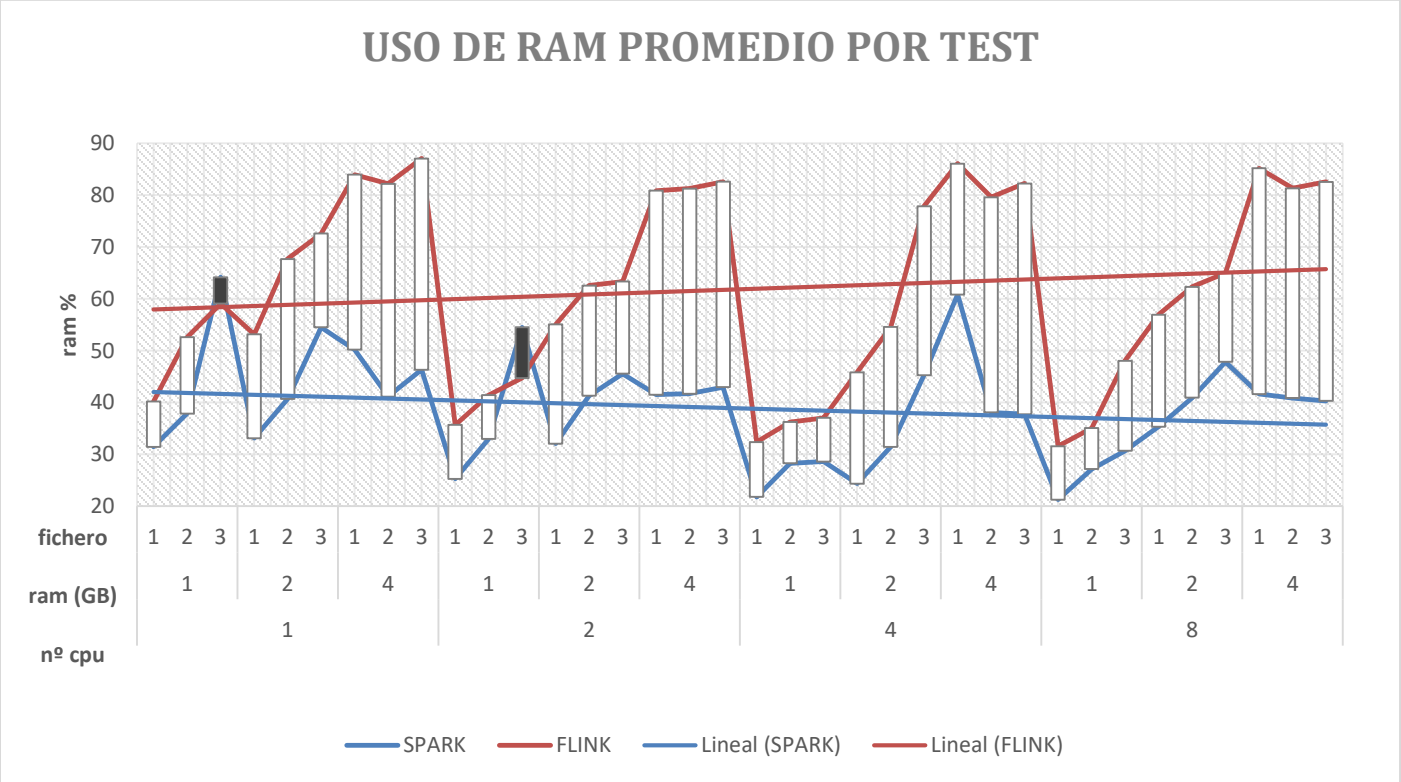


Gráfica XV K-Means: Saprk vs Flink - Tiempo

En la ejecución de estas pruebas, hemos tenido que aplicar un castigo a las herramientas que no terminen de ejecutar su algoritmo con los recursos asignados, por ello, la penalización que hemos considerado adecuada ha sido de 2000 segundos (20% del tiempo máximo en terminar una prueba).

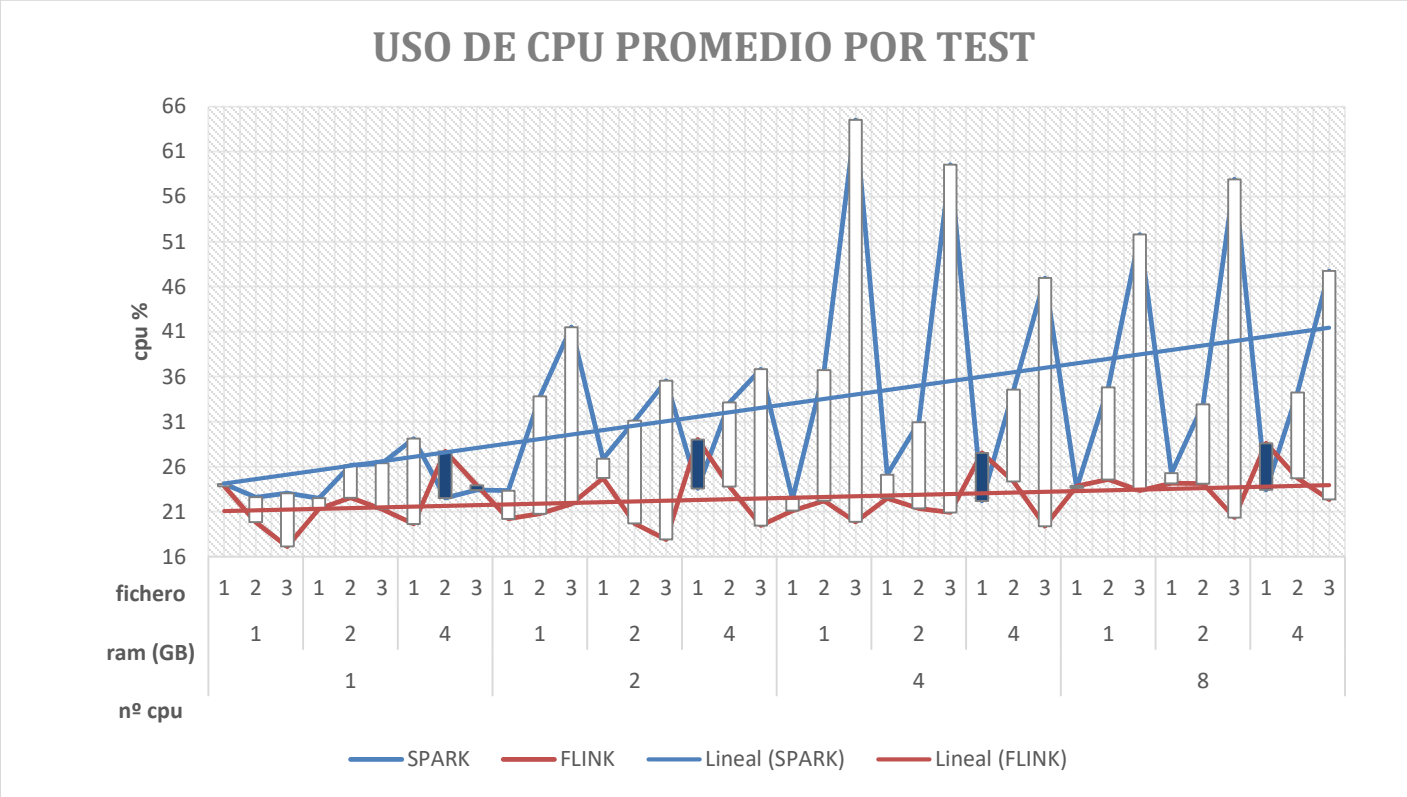
En este escenario Flink se comporta de manera muy superior a Spark, puesto que termina con éxito todas sus ejecuciones y en un tiempo mínimo. Spark por su parte sufre varios fallos de ejecución por falta de memoria: "**Spark java.lang.OutOfMemoryError: Java heap space**". La única manera de solucionar este fallo es aumentar la asignación de memoria **executor** (para más información del funcionamiento de Spark : Capitulo [2.3.2]).

En el consumo de memoria, observamos como Flink hace una gestión más adecuada de ella, pues utiliza toda la memoria que tiene a su disposición, mientras que Spark la administra de forma totalmente diferente.



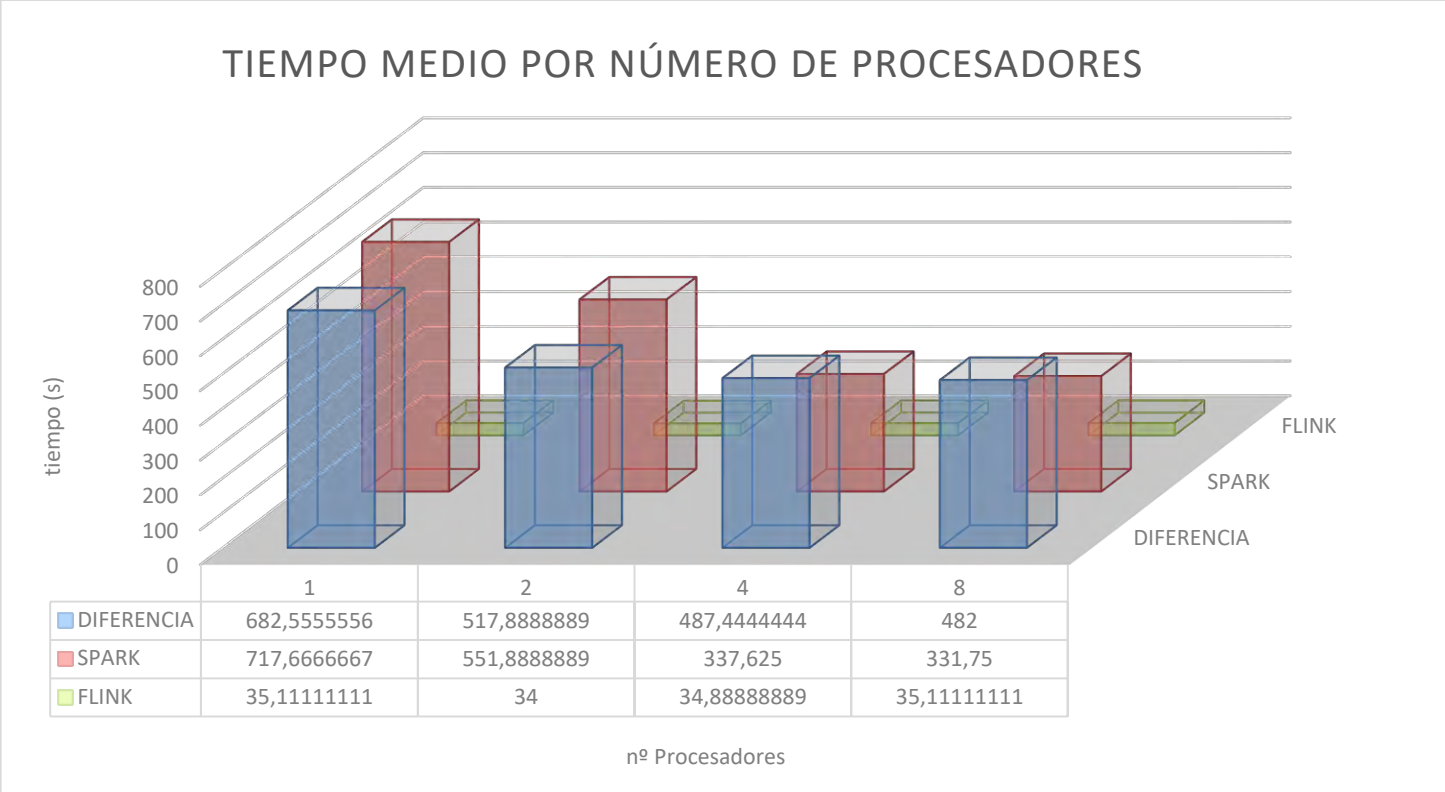
Gráfica XVI K-Means: Uso de CPU Promedio por Test

En la gráfica que vemos a continuación se observa el uso de CPU que consumen para procesar el fichero, observamos un comportamiento similar al ocurrido en PageRank, Flink permanece mucho más constante a lo largo de las pruebas mientras que Spark necesita hacer mucho más uso de CPU, esto es especialmente claro en el caso del fichero 3.



Gráfica XVII K-Means: Uso de RAM Promedio por Test

En el gráfico que muestra el tiempo medio por número de procesadores usados, podemos ver la diferencia de rendimiento entre Flink y Spark, es muy significantica debido a que Spark sufre varias penalizaciones por no terminar la ejecución de sus algoritmos y que Flink permanece constante entorno a los **35 segundos** de media por ejecución. Spark se comporta especialmente mal con un solo procesador, en este punto la diferencia entre las dos herramientas es de **682 segundos** de media.



	1	2	4	8
DIFERENCIA	682,5555556	517,8888889	487,4444444	482
SPARK	717,6666667	551,8888889	337,625	331,75
FLINK	35,11111111	34	34,88888889	35,11111111

Gráfica XVIII K-Means: Comparación de Tiempo por Nº de procesadores usados

4.2.7. Análisis

En este apartado vamos hacer un pequeño análisis por cada uno de los algoritmos ejecutados.

4.2.7.1. Word Count

RESUMEN DE RESULTADOS				
FRAMEWORK	**Nº CPU**	**RAM %**	**CPU %**	**TIEMPO (S)**
SPARK	1	40,84	18,99	233,56
	2	32,95	31,64	142,78
	4	33,76	50,71	104,75
	8	33,66	72,38	101,67
FLINK	1	67,28	19,50	289,44
	2	62,20	31,39	165,33
	4	60,34	51,37	101,25
	8	60,12	75,58	80,22
RECURSOS CONSUMIDOS - PROMEDIO				
SPARK		35,3	43,40	145,58
FLINK		62,48	44,46	158,38

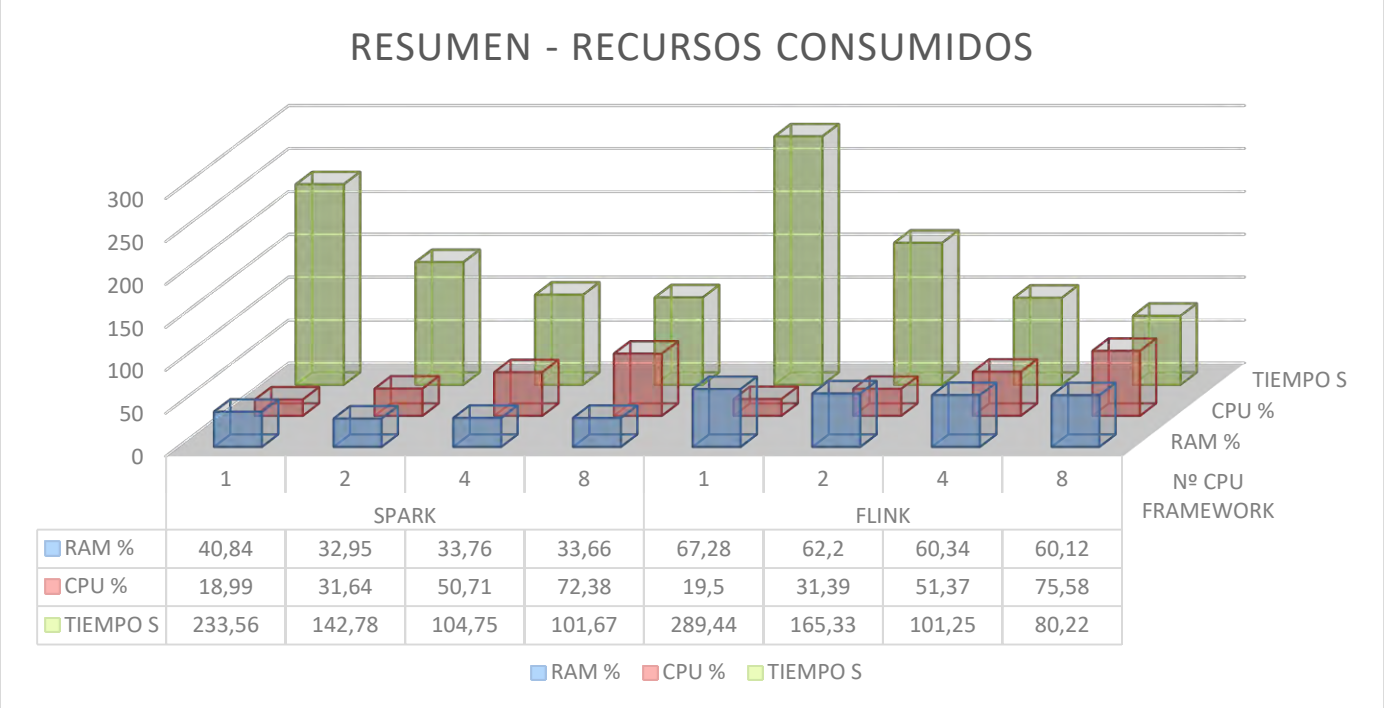
Tabla XXVII WordCount: Resumen de los datos obtenidos

El problema de Contar Palabras, es un escenario que se adapta muy bien a la arquitectura de **MapReduce** [11], Spark al combinar este enfoque junto las estructura de memoria RDD, esto debería permitir que los resultados fueran mejores que los de Flink que utiliza un enfoque algo distinto haciendo uso de las estructura **DataSet** [21], las cuales implementan una estructura **Map** (clave, valor), pero el **Reduce** (el agrupamiento de los datos) lo implementa de forma distinta.

Si observamos los resultados, vemos como Spark, el cual con pocos recursos se ha comportado peor que Flink, obtiene mejores datos. Pero al aumentar RAM y nº CPU Flink empieza a mejorar sus resultados hasta conseguir procesar los ficheros en un tiempo menor que Spark, el cual parece haberse **estabilizado en los 100 segundos**, pues la diferencia de ejecución entre 4 y 8 procesadores, no es significativa.

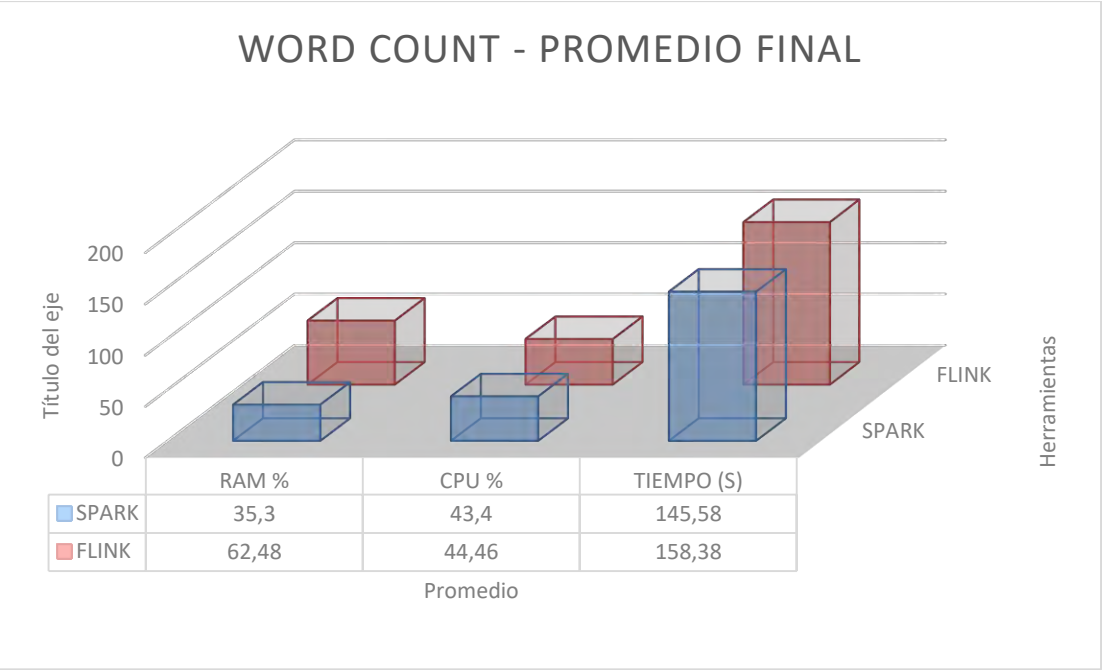
Por el contrario, Flink va mejorando sus resultados en cada Test, por lo que fijándonos en las líneas de tendencias del grafico [WordCount: Spark vs Flink - Tiempo] apostaríamos que Flink se comportará mejor con más cantidad de procesadores y memoria.

En el siguiente gráfico tenemos un resumen de las mediciones recogidas a lo largo de las pruebas realizadas en cada Frameworks. Aquí vemos muy claramente como tanto Spark como Flink mejoran el tiempo de ejecución con el aumento de CPU/RAM. En el consumo de CPU ambos realizan un consumo muy similar, pero en el caso de la memoria Flink consume entre un 20% y un 30% más que Spark.



Gráfica XIX WordCount: Resumen Recursos Consumidos

En el último grafico se aprecia mejor lo anteriormente comentado sobre el consumo de memoria, pues el promedio de memoria consumida en todas las ejecuciones por parte de Flink es de **27% mayor**, el consumo **de CPU es igual** y el tiempo medio que se obtiene en el procesamiento de la información es **8 segundos** más rápido Spark.



Gráfica XX WordCount: Promedio Final - Resumen

4.2.7.2. Page Rank

RESUMEN DE RESULTADOS				
FRAMEWORK	Nº CPU	RAM %	CPU %	TIEMPO S
SPARK	1	25,00	24,42	88,78
	2	31,58	31,74	79,44
	4	29,44	38,10	79,44
	8	28,20	36,86	71,67
FLINK	1	47,24	21,91	50,78
	2	60,36	21,93	54,56
	4	55,26	22,13	60,33
	8	54,43	24,00	63,33
RECURSOS CONSUMIDOS - PROMEDIO				
SPARK		28,55	32,78	79,83
FLINK		54,32	22,49	57,25

Tabla XXVIII PageRank: Resumen de los datos obtenidos

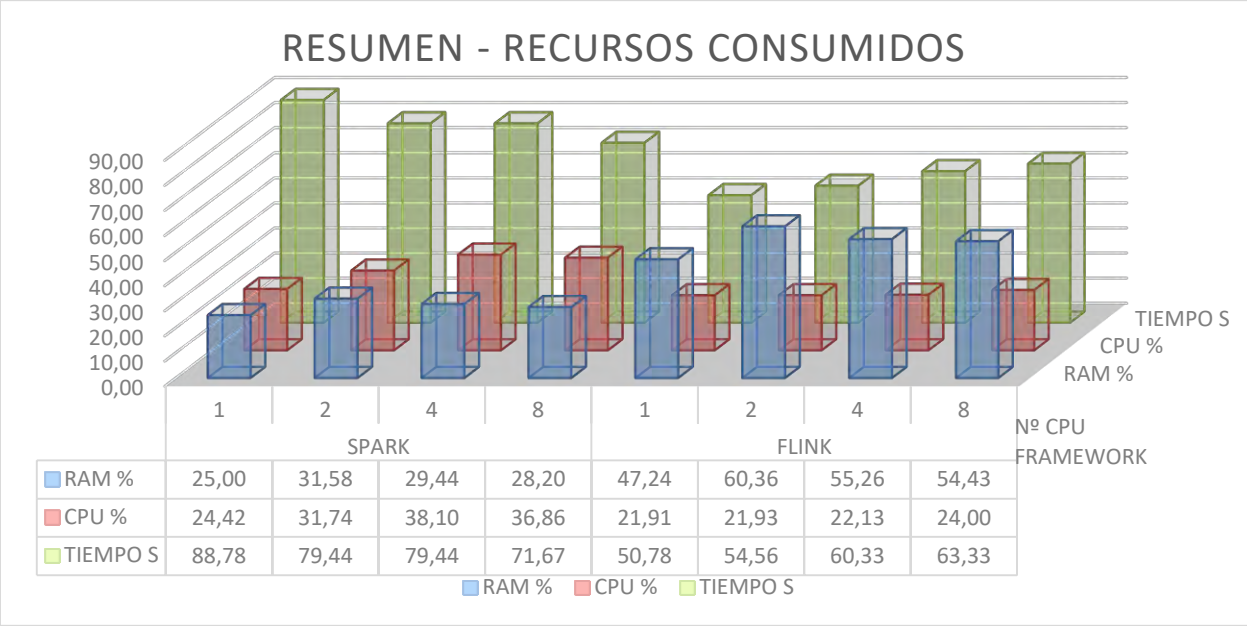
Este problema de Page Rank lo clasificamos como un problema de Grafos y por lo tanto utilizaremos las librerías **Gelly** de Flink y **GraphX** en Spark.

Si analizamos los resultados obtenidos, vemos como Spark mejora el tiempo de respuesta en función de los recursos que le asignemos, en cuanto a Flink, llama la atención que cuantos más recursos asignamos peor son los resultados que obtenemos. A pesar de ello sigue respondiendo mejor que Spark.

Esta singularidad la atribuimos la librería de grafos, la cual no se encuentre lo suficientemente madura (Recordemos que Flink nace en 2015) para realizar una computación particionada de los datos en distintos procesadores.

Analizando el consumo de CPU, encontramos que Spark hace un uso mayor mientras que Spark permanece prácticamente constante. Esto confirma que Flink no está haciendo uso de todos los procesadores que tenía a su disposición.

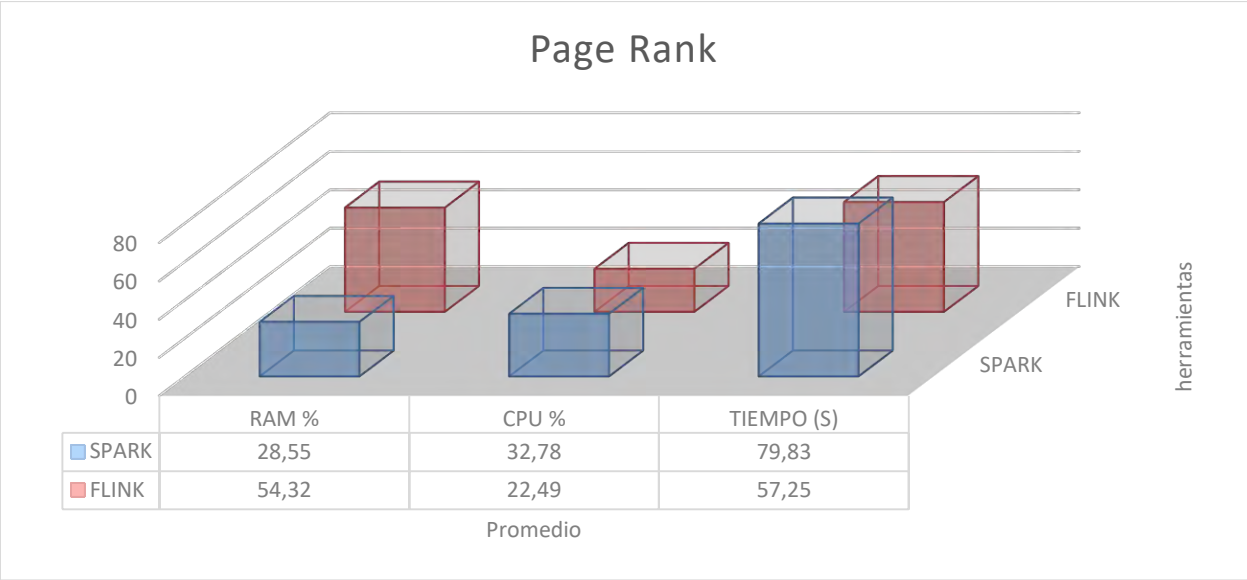
En cuanto al consumo de RAM, Flink sigue ocupando más memoria, este puede ser uno de los motivos por los que Flink tiene un tiempo de respuesta menor, pues a la vista de los datos obtenidos, Spark hace uso del **Swap** (Memoria Virtual) con mayor frecuencia que Flink. Esto último tiene bastante sentido si miramos la arquitectura de Spark, el cual separa los problemas en micro-trabajos y los guarda en estructura RDD las cuales va procesando en bloques.



Gráfica XXI PageRank: Resumen Recursos Consumidos

Por último, vemos una visión global de los resultados, se observa el mayor uso de memoria en Flink con casi un **30%** más.

En cuanto al tiempo medio global, Flink consigue terminar en **20 segundos** menos.



Gráfica XXII PageRank: Promedio Final - Resumen

4.2.7.3. K-Means

RESUMEN DE RESULTADOS				
FRAMEWORK	Nº CPU	RAM %	CPU %	TIEMPO S
SPARK	1	44,36	30,42	717,66
	2	39,73	42,87	551,88
	4	35,14	46,08	522,33
	8	36,20	46,68	517,11
FLINK	1	66,49	47,35	35,11
	2	60,81	46,77	34,00
	4	59,07	47,01	34,88
	8	60,87	47,55	35,11
RECURSOS CONSUMIDOS - PROMEDIO				
SPARK		38,85	41,51	577,25
FLINK		61,81	47,17	34,77

Tabla XXIX K-Means: Resumen de los datos obtenidos

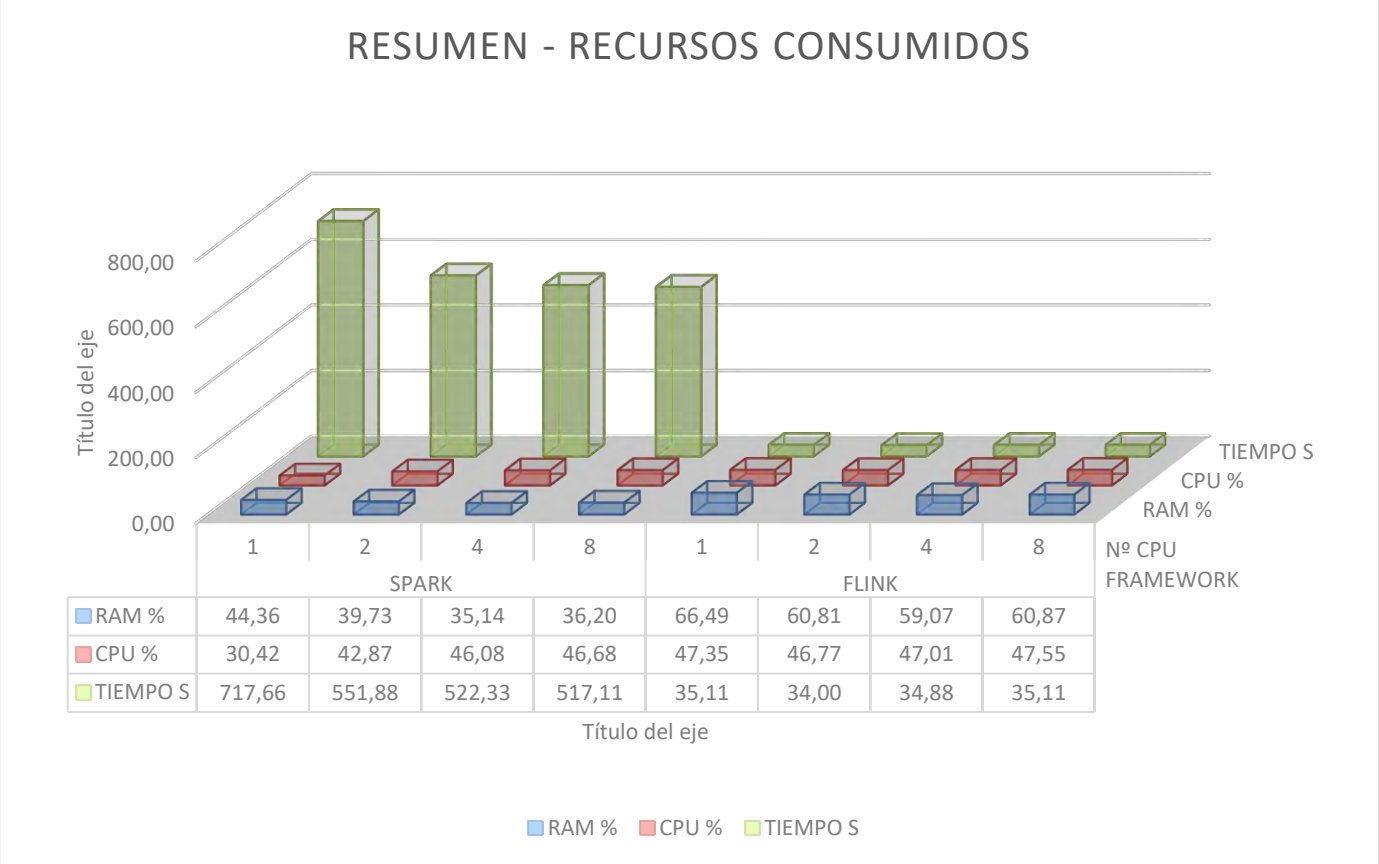
El algoritmo de K-Means, utiliza las librerías de Machine Learning de ambas herramientas y estas a sus vez utilizan las estructuras que se mencionan en el algoritmo de Word Count [Capítulo: 904.2.7.1], este caso es interesante de analizar por el comportamiento tan dispar que han tenido ambas herramientas.

Spark ha tenido bastantes problemas a la hora de procesar los ficheros, pues 3 ocasiones tuvo problemas de memoria lo que provoco que no pudiera terminar de ejecutar las evaluaciones y por consiguiente tener una penalización de tiempo.

Observando los resultados, vemos que el consumo de CPU ha sido muy igual entre las dos herramientas de análisis, por el contrario, el uso de RAM ha sido más elevado en Flink, pero sin llegar a usar toda la memoria que tenía disponible. En el caso de Spark se observa un patrón a lo largo de todas las pruebas y es su gestión de memoria, pues a pesar de tener memoria disponible para utilizar no es capaz de hacer uso de ella.

El tiempo de respuesta de Flink permanece constante durante todas ejecuciones, esto nos indica que podría procesar archivos más complejos donde realmente podríamos ver una reducción de tiempo.

Spark a pesar de reducir su tiempo de respuesta, vemos que llega a su límite de mejora con 4 y 8 procesadores y no consigue ninguna mejora significativa.

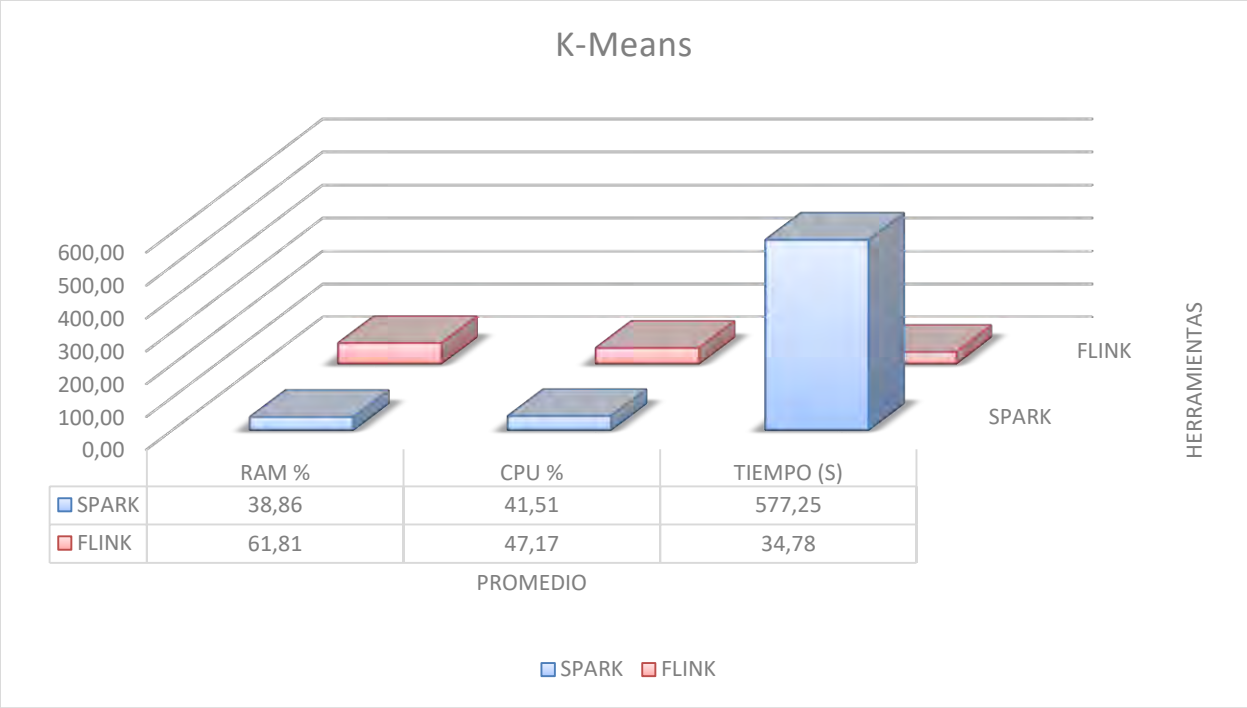


	SPARK				FLINK			
Nº CPU	1	2	4	8	1	2	4	8
RAM %	44,36	39,73	35,14	36,20	66,49	60,81	59,07	60,87
CPU %	30,42	42,87	46,08	46,68	47,35	46,77	47,01	47,55
TIEMPO S	717,66	551,88	522,33	517,11	35,11	34,00	34,88	35,11

Gráfica XXIII K-Means: Resumen Recursos Consumidos

Analizando este último gráfico de los resultados vemos la diferencia de tiempo entre ambas herramientas, obteniendo una diferencia de **543 segundos** a favor de Flink.

El consumo de memoria que obtiene Flink es superior al de Spark en un **23%** algo que se ha convertido en un dato característico de las pruebas.



Gráfica XXIV K-Means: Promedio Final - Resumen

4.2.8. Informe de conclusiones

A continuación, vamos a ofrecer las conclusiones a las que hemos llegado después de analizar los resultados obtenidos en las distintas pruebas y algoritmos ejecutados. También hablaremos de las sensaciones que hemos tenido al trabajar con estas herramientas.

Empezaremos hablando de Apache Storm, del cual no tenemos datos sólidos debido al tiempo desproporcionado que ha tardado en procesar los algoritmos y que hacía imposible su comparación. En futuros trabajos se podría montar la arquitectura aquí diseñada en un entorno distribuido para comprobar su rendimiento.

4.2.8.1. Primeras Impresiones

En referencia a Spark y Flink, empezaremos hablando de las sensaciones que trasmiten cuando empiezas a trabajar con ellos, desde el punto de vista de una persona que no había tenido relación con ninguna herramienta parecida.

La primera impresión que tienes es que Spark es mucho más sencillo de utilizar y de ponerlo a funcionar, pues con poco esfuerzo puedes ejecutar los algoritmos de ejemplo. Por el contrario, Flink, parece más difícil de utilizar, es más engorroso y no parece que vayas a terminar de hacerlo funcionar.

Esto principalmente se debe a que es Spark hay más documentación, pues se trata de un proyecto con más recorrido y que más gente ha usado.

Pero una vez empiezas a entender cómo funciona Flink, sorprende lo fácil de configurar y la buena gestión de memoria que implementa facilita enormemente la ejecución de los algoritmos, pues con esta herramienta no hemos tenido un solo problema de memoria.

Si hablamos de cambios, Spark mejoró el enfoque MapReduce combinándolo con RDD, pero no pienso que sea una revolución en absoluto, recordemos que Spark tiene mucha de la esencia de Apache Hadoop, pero con un enfoque un poco distinto. Por el contrario, basándonos en los resultados, vemos como Flink sí que supone una un gran paso.

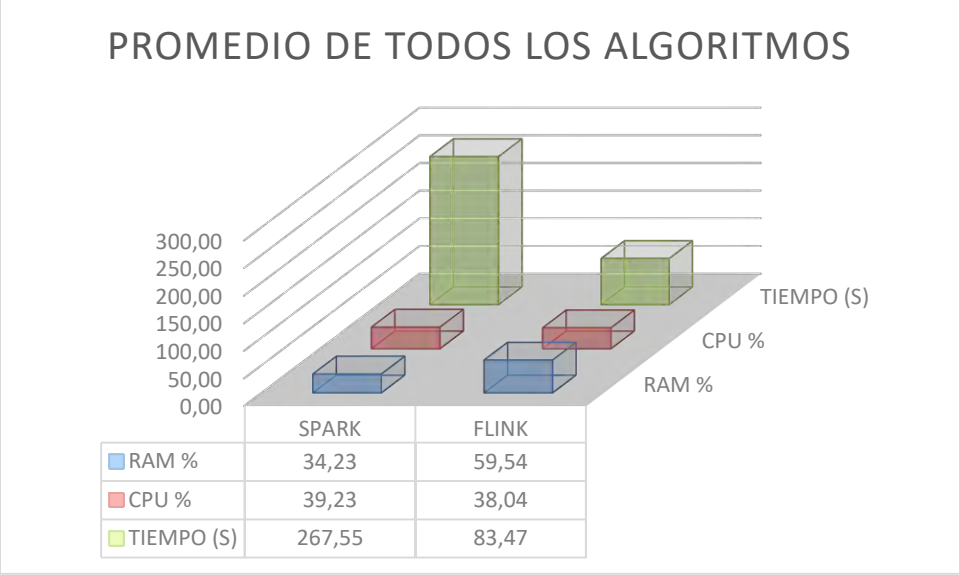
4.2.8.2. Barreras de Entrada

La posibilidad de empezar teniendo contacto con Spark mediante Python (Al ser un leguaje interpretado facilita su ejecución) facilita el empezar a trabajar, pues permite de una forma muy ágil, copias algunas líneas de código y ejecutarlo, intentar hacer esto en java es una utopía, pues a pesar de que todos "hemos trabajado" con el leguaje, cuando se trata de un proyecto grande el cual se gestiona con **maven**, tienes la sensación de no haber escrito una sola línea de código y tener la sensación de no saber cómo avanzar. Flink incorpora también la posibilidad de ejecutar programas en Python, pero no funciona tan bien como en Spark.

4.2.8.3. Resultados de los Experimentos

Una vez hemos ejecutado los algoritmos en las dos herramientas mientras monitorizábamos la evolución del hardware hemos llegado a las siguientes conclusiones.

Basándonos en los experimentos realizado y los resultados obtenidos, la arquitectura que plante Spark, supone una evolución en las herramientas de análisis de datos. En general Todos los algoritmos ejecutados en Flink se han ejecutado más rápido y con menos complicaciones.



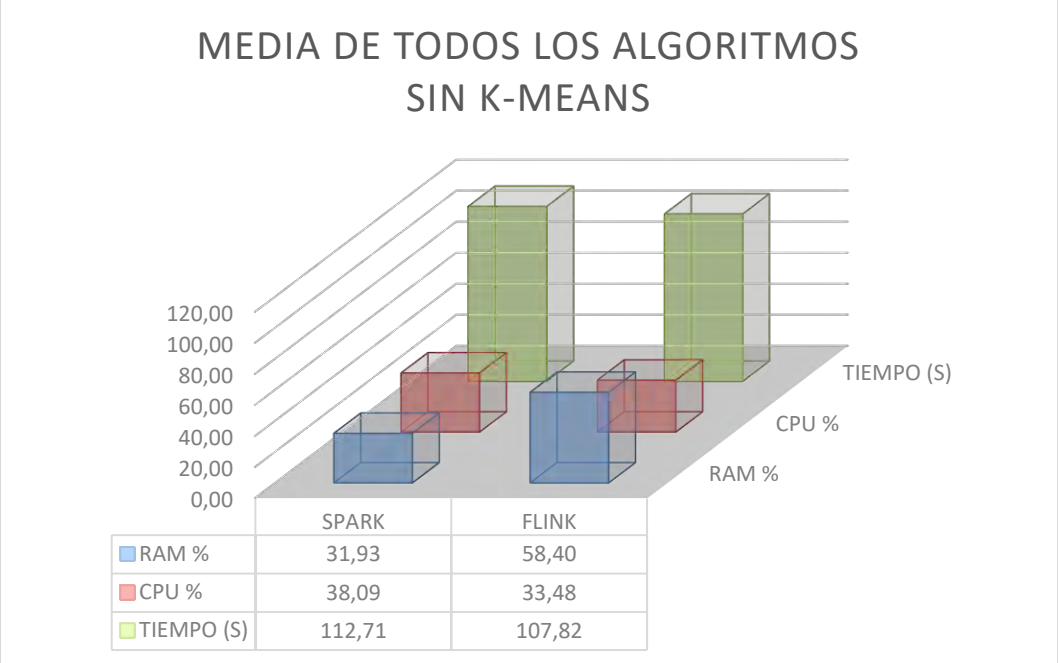
Gráfica XXV Conclusiones: Promedio de las mediciones

El gráfico anterior, nos muestra una perspectiva global, cierto es que la media no es el mejor estimador estadístico, consideramos que es este escenario lo podemos utilizar pues dentro de los distintos experimentos no se hay resultados extremos que puedan desvirtuar esta conclusión.

Como hemos visto a lo largo de todas las pruebas, Spark ocupa más en memoria, entorno a un 20% más respecto a Spark.

En cuanto al uso de CPU, las diferencias no son significativas, por lo que podríamos decir que ambas tienen un rendimiento muy parecido en el uso de CPU.

Y analizando el tiempo en completar los algoritmos, Flink completa las tareas en un tiempo menor. Es cierto que la mayor diferencia de tiempo viene dada por K-Means, pero si analizamos los resultados si tener en cuenta este último algoritmo



	SPARK	FLINK
RAM %	31,93	58,40
CPU %	38,09	33,48
TIEMPO (S)	112,71	107,82

Gráfica XXVI Conclusiones: Sin K-Means

Se observa que los valores de CPU/RAM se mantienen constantes con respecto, a los resultados con todos los algoritmos. Donde realmente hay un cambio, es en el tiempo medio de respuesta el cual pasa de existir una diferencia de 184 segundos a apenas 5

segundos, pero si analizamos uno por uno los casos vemos que existe una tendencia de progresión y mejora en Flink.

Por lo que concluimos que Flink es una herramienta que implementa una mejora tangible dentro de las herramientas de procesamiento de información o Herramientas de Big Data.

CAPÍTULO 5

5 Planificación y Presupuesto

5.1 Planificación

A lo largo de este apartado se va a explicar las fases en las que se ha dividido la realización del trabajo de fin de grado.

5.2 Definición de Tareas

La elaboración del presente proyecto se ha dividido en tres fases, que se describen a continuación:

- Fase 1: Análisis.
  - Duración**: 22 días**

Para comenzar la realización del trabajo, se procede a realizar un análisis de los principales componentes que van a intervenir.

- Planteamiento del escenario.
  - Análisis de todos los componentes que van a intervenir.
- Análisis de los Frameworks BIG DATA.
  - Cada Framework tiene funciona de una forma en concreto, buscamos sus similitudes, así como sus diferencias.
- Análisis de Microservicios a utilizar.
  - Analizamos la mejor forma de aislar los Frameworks con el menor impacto para el S.O. anfitrión
- Búsqueda Repositorios de Datos.
  - Realizamos una comparativa entre distintos repositorios de datos para alimentar a las herramientas.
- Análisis de medición de recursos

    - Teniendo en cuenta las características del Hardware sobre el que se van a realizar las ejecuciones, es necesario que la medición de uso de CPU, RAM y tiempo sea lo más preciso posible.
- Análisis de pruebas a realizar.
    - Con el fin de poder analizar correctamente los Frameworks es necesario realizar pruebas que comparen la API de cada Framework.
- Análisis BBDD.
    - La BBDD se usará para almacenar las pruebas realizadas, así como sus resultados, realizamos un análisis para diseñar.
- Análisis de interfaz Front-end.
    - Realizamos un análisis de todos los elementos que se van a representar, cómo son los gráficos o el estado de las pruebas.

➢ Fase 2: Desarrollo.
    - Duración: **73 días**

Una vez hemos realizado el análisis previo, procedemos a realizar el desarrollo de la herramienta que permitirá lanzar pruebas de ejecución sobre los distintos Frameworks para comparar cual es más óptimo dependiendo de los factores de ejecución (algoritmo y fichero de datos)

- Desarrollo BBDD.
    - Creamos las tablas que almacenan las pruebas.
- Configuración de imágenes Docker.
    - Apache Spark
    - Apache Flink
    - Apache Storm
    - Realizamos la instalación de cada una de las herramientas tomando como base **Ubuntu 14.04 LTS,** se instalan todos los componentes necesarios para el correcto funcionamiento.
- Compilamos Programas Test.
    - Compilamos la API de ejemplos para poder ejecutarla. Se utiliza Maven para empaquetar todos los ficheros.
- Desarrollo de Bundle de Symfony.
    - Toda la herramienta se realizará sobre Symfony un Frameworks web que nos ahorra tiempo en el desarrollo de aplicaciones WEB.
- Desarrollo vistas Front-End.
- Implementación Entidades BBDD a POO (ORM).

- Implementación WS RestFul
- Implementación Sistema Benchmark
- Implementación Automatización del Benchmark

➢ Fase 3: Documentación.
  - Duración: **33 días**.

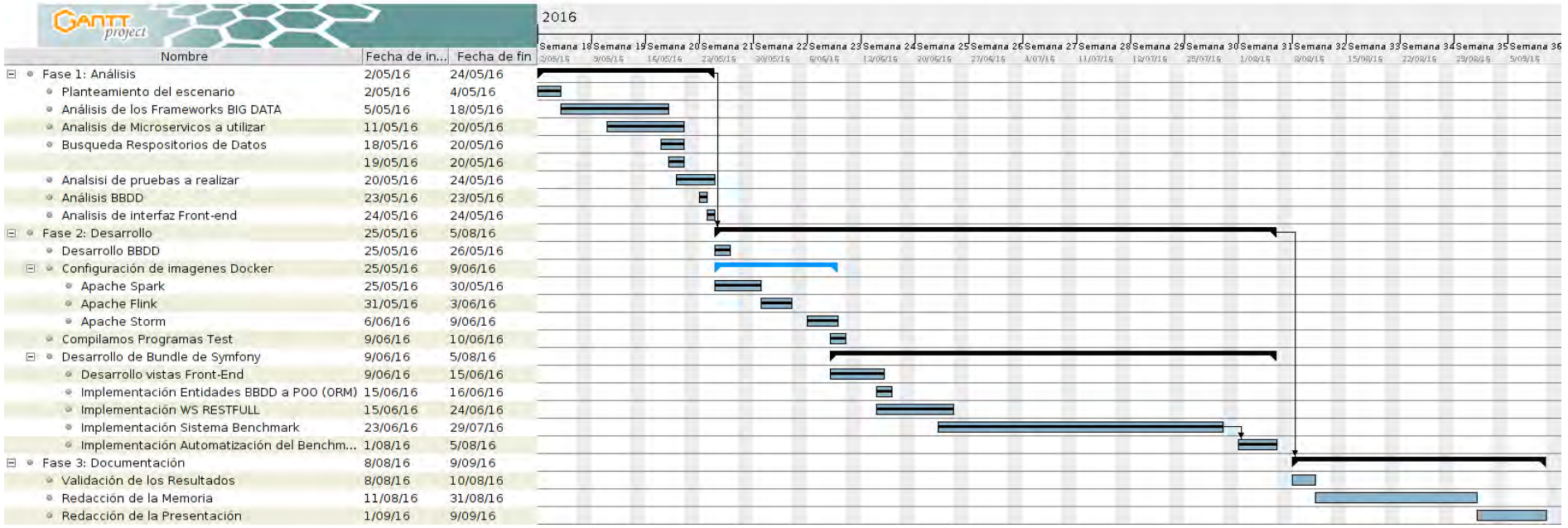
Por último, se realiza la validación de los resultados obtenidos y se comienza la redacción de la presente memoria, así como la presentación del proyecto.

- Validación de los Resultados
- Redacción de la Memoria
- Redacción de la Presentación

La planificación está dividida en tres fases, las cuales representan un camino crítico. Un retraso en estas tareas podría suponer un retraso en el desarrollo del proyecto. Dentro de la fase de desarrollo, encontramos varias tareas críticas como es la **configuración de imágenes Docker** y de **Desarrollo de Bundle.**

5.3 Diagrama de Gantt

Duración del Proyecto
4 Meses



Nombre	Fecha de in...	Fecha de fin
Fase 1: Análisis	2/05/16	24/05/16
Planteamiento del escenario	2/05/16	4/05/16
Análisis de los Frameworks BIG DATA	5/05/16	18/05/16
Analisis de Microservicos a utilizar	11/05/16	20/05/16
Busqueda Respositorios de Datos	18/05/16	20/05/16
	19/05/16	20/05/16
Analsisi de pruebas a realizar	20/05/16	24/05/16
Análisis BBDD	23/05/16	23/05/16
Analisis de interfaz Front-end	24/05/16	24/05/16
Fase 2: Desarrollo	25/05/16	5/08/16
Desarrollo BBDD	25/05/16	26/05/16
Configuración de imagenes Docker	25/05/16	9/06/16
Apache Spark	25/05/16	30/05/16
Apache Flink	31/05/16	3/06/16
Apache Storm	6/06/16	9/06/16
Compilamos Programas Test	9/06/16	10/06/16
Desarrollo de Bundle de Symfony	9/06/16	5/08/16
Desarrollo vistas Front-End	9/06/16	15/06/16
Implementación Entidades BBDD a POO (ORM)	15/06/16	16/06/16
Implementación WS RESTFULL	15/06/16	24/06/16
Implementación Sistema Benchmark	23/06/16	29/07/16
Implementación Automatización del Benchm...	1/08/16	5/08/16
Fase 3: Documentación	8/08/16	9/09/16
Validación de los Resultados	8/08/16	10/08/16
Redacción de la Memoria	11/08/16	31/08/16
Redacción de la Presentación	1/09/16	9/09/16

Ilustración XXIII Diagrama Gantt

5.4 Presupuesto

En el siguiente apartado se desglosan los costes, tanto materiales como de personal, asociados a la realización de este proyecto.

5.4.1. Costes de Personal

En la realización de este proyecto se han implicado varias personas, el tutor y el desarrollador principal del estudio. Para obtener el coste del trabajo de cada uno de ellos se ha aplicado como coste por hora el salario medio a las horas de trabajo aportadas por cada uno de los componentes del equipo:

Categoría Profesional	A	B
Horas Semanales	15	29
Salario Mensual (€)	4000,00	2000,00
IRPF (20%) (€)	800,00	400,00
Seguridad Social (40%) (€)	1600,00	800,00
Coste Semanal (€)	1600,00	800,00
Coste Mensual (€)	6400,00	3200,00
Horas Totales Dedicadas	45	116
Coste Total Horas (€)	106,67	27,59
Suma Total Sueldos (€)	8000,00	

Tabla XXX Costes en Recursos Humanos

Categoría Profesional	Descripción
A	Jefe de Proyecto
B	Encargado de Análisis / Desarrollo

Tabla XXXI Descripción de las categorías Profesionales

5.4.2. Costes de Material

Dentro de este apartado se muestran los costes asociados tanto a la adquisición de materiales físicos necesarios para llevar a cabo el proyecto como las licencias de los programas utilizados en el mismo.

Descripción	Coste(€)	Amortización	Periodo Depreciación	Coste Imputable (€)
Ordenador Sobremesa i7 6GB RAM 256 SSD	1000,00	26%	12 Meses	86,67
2 x Monitores TFT 21" (140€ u.d.)	280,00	26%	12 Meses	24,26
HP LaserJet Pro MFP M127fw	211,00	26%	12 Meses	18,26
1 x Toner para la impresora				189,36
Paquete de 500 Folios				2,00
Caja Bolígrafos BIC				9,99
Repositorio de Código GitHub				0
Repositorio Docker				0
PACK (Luz, Agua, Internet) (49€/mes)				196,00
Coste Total Gastos				**526,54**

Tabla XXXII Costes en Materiales

El coste imputable de los materiales que sufren una depreciación con el paso del tiempo se obtiene mediante la siguiente fórmula:

$$\text{Coste Imputable} = \frac{\text{Duración Proyecto}}{\text{Período de depreciación}} \times \text{Coste} \times \text{Amortización}(\%)$$

5.4.3. Costes Totales

Ahora se va a proceder a la unificación de ambos tipos de coste para obtener el precio del total del proyecto.

COSTE TOAL DEL PROYECTO		
Costes Directos (€)	**Personal**	**Material**
	8000,00	526,54
Costes Indirectos (€)	1900,00	
Beneficio 20% (€)	2122,00	
IVA 21% (€)	26435,24	
Total Proyecto (€)	**15.183,73**	

Tabla XXXIII Costes totales del Proyecto

El coste final del proyecto es de: (**15.183,73 €**) **Quince mil ciento ochenta y tres euros con setenta y tres céntimos de euro.**

CAPÍTULO 6

6 Trabajos Futuros

6.1 Producto

En este apartado vamos repasar los objetivos que inicialmente se plantearon, con el fin de comprobar que todos los requisitos se cumplen.

El primer objetivo que se deseaba conseguir con la realización de este proyecto, era la de diseñar e implementar un Benchmark que permitiera automatizar la ejecución de pruebas junto a un sistema que midiera el impacto que tenía la ejecución de los algoritmos en términos de rendimiento.

Este punto ha sido ejecutado y completado sin incidencias, pues la aplicación desarrollada dispone de una BBDD donde se almacenan las pruebas creadas y permite ejecutarlas una por una a la vez que mide la evolución de uso de CPU y Memoria RAM hasta que termina la ejecución del algoritmo, lo que nos permite medir el tiempo de ejecución del algoritmo.

El segundo objetivo que se planteó fue la de comparar cada una de las tres herramientas con distintos algoritmos y ficheros.

Este segundo punto ha sido ejecutado y completado sin incidencias, pues la aplicación mide el estado del sistema antes de iniciar la prueba, durante la ejecución y al finalizar. Posteriormente todas las mediciones son mostradas en la interfaz en forma de tabla y graficando la evolución de la prueba de forma individual.

El tercer objetivo que se buscaba era comparar cuál de las tres herramientas se comportaba mejor, en función del escenario elegido (algoritmo y fichero de datos)

Este tercer punto ha sido ejecutado y completado sin incidencias, pues la aplicación permite la ejecución de los tres Frameworks tomando el mismo algoritmo y el mismo fichero de datos. Posteriormente nos enfrenta las mediciones de las tres

herramientas, en formato de gráfica, para que se pueda analizar qué herramienta funciona mejor en función del caso planteado. De forma adicional, si queremos, tenemos la posibilidad de descargar los datos en formato CSV de las mediciones de cada prueba de forma individual.

6.2 Proceso

En este apartado revisaremos los problemas de planificación que hemos sufrido en el desarrollo de este Trabajo Fin de Grado o TFG.

Los principales problemas de planificación que hemos sufrido han sido causados a estimar mal el tiempo necesario en realizar ciertas tareas, las cuales mencionamos a continuación:

- Instalar y configurar las herramientas de análisis Big Data.
- Compilar los programas dentro de las herramientas usando Maven.

Estos retrasos nos han servido para aprender a planificar de mejor las tareas y nos permitirá en futuras planificaciones, añadir tiempo extra de desarrollo en cada fase para tener margen de maniobra y evitar en la medida de lo posible los retrasos.

6.3 Personales

En este punto procederemos a repasar las asignaturas del Grado en Ingeniería Informática que sido útiles en el desarrollo de la aplicación y de qué manera han influido y por otro lado los conocimientos que hemos adquirido por cuenta propia y que no han sido propios del Grado.

Las asignaturas que hemos visto a lo largo del Grado y que han ayudado en el desarrollo de este trabajo han sido:

- **Programación** y **Estructura de Datos y Algoritmos**: Por plantar las bases de programación que han permitido desarrollar el programa.
- **Técnicas de Búsqueda y uso de la Información**: Nos enseñó a como buscar información, libros de la biblioteca …
- **Técnicas de Expresión Oral y Escrita:** Esta asignatura nos ayudó a expresarnos de forma correcta a la hora de elaborar documentos.

- **Estructura de Computadores** y **Arquitectura de Computadores:** Adquirimos habilidades de programación y entendimos cómo se gestiona el hardware, muy útil para el sistema de medición de recursos usados.
- **Ficheros y Base de Datos:** Gracias a esta asignatura hemos podido diseñar una BBDD donde almacenar los datos de las pruebas.
- **Ingeniería del Software** y **Dirección de Proyectos de Software:** Estas asignaturas han sido muy importantes para la planificación, obtención de requisitos y desarrollo de la memoria.
- **Diseño de Sistemas Operativos:** Con esta asignatura aprendimos aspectos muy importantes en el diseño de sistemas operativos, aspectos que son extrapolables u otros tipos de diseño de software como el de este proyecto.
- **Tecnologías Informáticas para la Web:** Puso las bases para poder gestionar una aplicación web al completo.

A continuación, analizamos los conocimientos adquiridos por nuestra propia cuenta:

- Ha sido necesario aprender cómo funciona Docker y como instalarlo y configurarlo.
- Aprender como Instalar y configurar Apache Storm Apache Spark y Apache Flink.
- Entender cómo se debían ejecutar los algoritmos de los tres Frameworks que hemos analizado.
- Aprender a compilar los algoritmos que queríamos implementar utilizando Maven.
- Aprender a usar Symfony para implementar el Dashboard e interactuar con la BBDD.
- Enfrentarnos al desarrollo de un proyecto completo, con sus respectivas complicaciones y que debíamos aprender a superar.

6.4 Trabajos Futuros

Vamos a detallar algunas mejoras que se podrían introducir al proyecto desarrollado:

I. Preparar las imágenes Docker para que el Benchmark se ejecute en un clúster real con un gran número de equipos.
II. Añadir una interfaz que permita comparar pruebas que hemos lanzado de forma individual, con la misma interfaz que se muestran las pruebas comparativas.
III. Permitir la carga de datos de evaluación desde internet de forma dinámica (en el momento de lanzar la prueba, procede a descargar los datos)
IV. Añadir un modo de prueba que ejecute los 3 Frameworks al mismo tiempo en entornos virtualizados e independientes.
V. Ampliar los algoritmos utilizados en la evaluación de las herramientas.

CHAPTER 7

7 English Competitions

This annex contains the English competitions, required for finishing the Bachelor´s degree in Computer Science and Engineering at Carlos III University.

First, we will introduce the project objectives, the principal motivation, and the structure that the project follows. Then, we will show the results of the test.

Finally, we will present the project conclusions and the future lines.

7.1 Introduction

This chapter contains a formal presentation of the document. This chapter describes, in a general form, the principal motivation that has promoted the project creation, as well as, the objectives that it targets and the project structure.

7.1.1. Motivation

We generate a lot of information that is becoming difficult to analyze. In 1880 they are beginning to have trouble calculating the United States Census, because it took 8 years to process all the information. This caused them to begin to look for better ways to process the data successfully obtained.

Advancing in time to 1941, he starts talking about the explosion of information referring to the difficulty posed manage so much data.

Edgar F. Codd in 1970 published an article where he talks of the relational database model, this will allow access to information much faster and reducing the complexity that existed at that time.

Today, routine transactions data, access bank accounts, using credit cards, hotel reservations, make purchases over the Internet use based on the theory of relational database structures.

As time passed, the speed with which information is created increased. Technological advances allow companies to start working with the data warehousing business to make decisions, what it is today known as Business Intelligence.

But it was not until the 90s World Wide Web appears and shoots technological development and the information generated began to pile up.

Peter Lyman and Hal R. Varian UC Berkeley published the first study to quantify in terms of computer storage, the total amount of new and original information created in the world

a year. The study, titled How Much Information?, was completed in 1999, a year in which the world produced about 1.5 exabytes (EB) of information.

1 EB = 1 073 741 824 GB = 10^9 GB

In 2006 Hadoop is created, allowing you to work with petabytes of data (
1 PB = 1048576 GB). He used of MapReduce and GFS (Google File System) Google as the basis for its development. This enabled distributed parallel processing of huge amounts of information [1].

In late 2011 it begins to develop Apache Storm in mid-2012 Apache Spark and finally in June 2015 Apache Flink appears. These frameworks are intended to be an improvement in Hadoop and thus able to analyze and process more information in less time and more efficiently.

7.1.2. Objetives

The main objective of this Bachelor's degree will be:

I. Designing a Benchmark to automate testing and measuring the performance of each of the tools.
II. Check how each tool behaves in the different scenarios that encounter.
III. Check the performance of the same algorithm in the different tools.

7.1.3. Structure

This document contains six chapters. Next, a brief introduction of each:

- **Chapter 1: Introduction.** Exposes the motivation and objectives of the thesis, document structure and the structure of memory.
- **Chapter 2: State of the Art.** Make a small introduction to the concept of BIG DATA and an analysis of the Frameworks analyzed.
- **Chapter 3: System Overview.** The tools used and explains why the choice in this project.
- **Chapter 4: Evaluation and Results**. Evaluation system developed and the results obtained.
- **Chapter 5: Planning and budget.** This section explains the phases in which the project is planned, tasks and the budget needed to carry out.
- **Chapter 6: Future work**. Sets out the proposals for possible future developments.

7.2 Evaluation

7.2.1. Algorithm Selection

This section will explain what have been the algorithms that evaluate and how they work.

7.2.1.1. Word Count

This algorithm as its name suggests is to count words. This approach is known as the "Hello World" programming languages in data analysis tools.

This algorithm receives a file or source of information which is obtained words, which are counted. Despite how simple it seems this implementation we can give a good indication of how good or bad is the tool, it has to deal with large amounts of information while the organized and processed. This algorithm fits well with the philosophy of MapReduce [11]

7.2.1.2. Page Rank

The PageRank algorithm [19] PR onwards, is one of the methods that Google used to determine the order of the results displayed in your internet browser. The original article "The Anatomy of a Large-Scale Hypertextual Web Search Engine" can be found on the Web http://infolab.stanford.edu/~backrub/google.html.

The algorithm PageRank assign a value to all websites, this value is higher if there are pages that link to you.

The general formula that describes the algorithm is as follows:

$$PR(A) = (1 - d) + d \sum_{i=1}^{n} \frac{PR(i)}{C(i)}$$

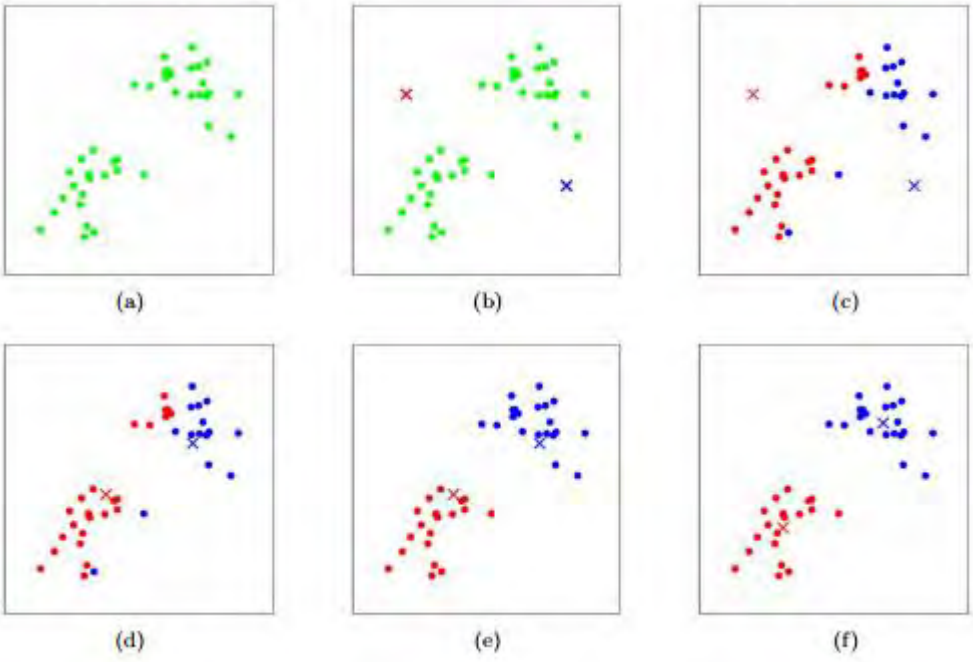
With this algorithm we can compare different models of memory management (structures used) and partitioning into subproblems in order to accelerate the calculation of the final solution.

7.2.1.3. K-Menas

This clustering algorithm, which partitions a group of n observations in K groups where each observation or is associated with Ki group, which has the closest average value.

$$J = \sum_{j=1}^{K} \sum_{i=1}^{n} \left\| X_i^{(j)} - C_j \right\| \text{^2}$$

In the next picture we see how the algorithm works:



As you can see, we start from a set of data, which we classify. The first step is to choose K centroids represented by X, usually these centroids can take observations.

Then we began to iterate and compare the remaining observations NK, where N is the total number of observations and K the number of elected centroid, and calculate the distance between the remaining observations and selected centroids assigned to each observation the centroid Ki more close.

Once all observations are assigned a centroid, which generates K groups or cluster, compute the new centroid of each group, and for this we calculate the average of each group and this will be the value of the new centroid.

This process is repeated until the distance values hardly change.

7.2.2. Metrics Selecting

This section will analyze the metrics that we used to measure performance offered by each tool of analysis. The indicators that we will analyze are:

- **Time it takes the algorithm to process the data:** This indicator seems to be the most representative for the most expensive resource and which we will always want to minimize.
- **CPU use and Evolution:** This indicator will measure us at every moment what are the moments of computational load, at what point are produced and how they are managed, so we plot the evolution and see a quick comparison**. This indicator gets the average CPU usage between all cores**.
- **Use and Evolution RAM:** This third indicator will inform us of the use of memory that is doing the analysis tool and how different data structures using occupy more or less memory. It will facilitate comparison of memory management that makes each tool.

7.2.3. Physical Enviroment Tuning

In paragraph [4.1] of this document, you can get an overview of the physical architecture on which have executed the evaluation of different algorithms. By budget limitations, these tests have to be done locally but the nature of software architecture allows easily export it to a model of distributed computing (everything is ready, only be necessary to specify a list of IP where distributing the calculation).

7.2.4. Analytics providers Tuning

We will perform different configurations of tools to evaluate them, the restrictions imposed on the problem are as follows:

I. RAM Constraints:
   a. 1 GB.
   b. 2 GB.
   c. 4 GB.

II. CPU Constraints:
   a. 1 CPU Core.
   b. 2 CPU Core.
   c. 4 CPU Core.
   d. 8 CPU Core.

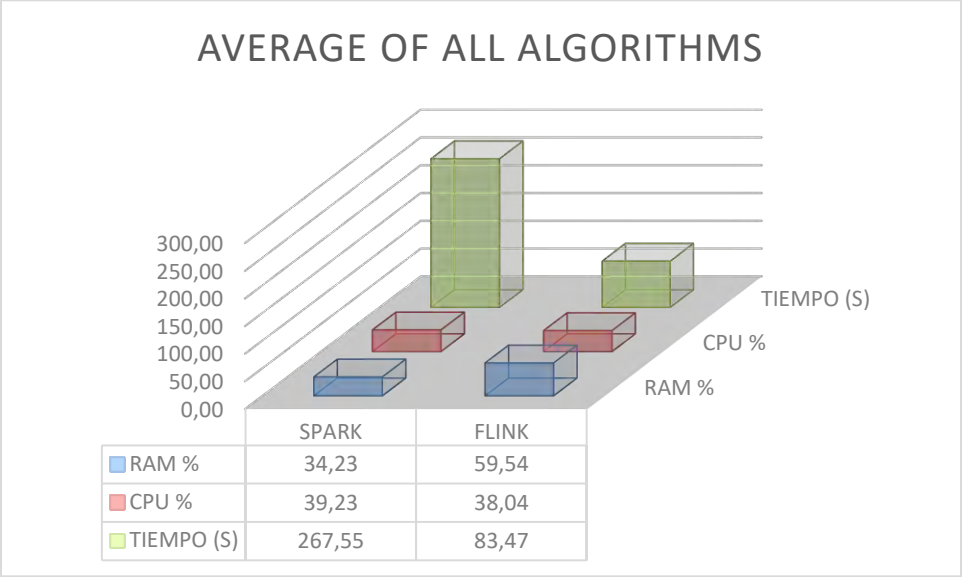
III. Data Constraints:
   a. File 1: Smaller
   b. File 2: Medium Size
   c. File 3: Large.

We generate 36 different configurations for each algorithm and analysis tool to evaluate. To learn how to set up the tools, we can go to chapter **[3.4.4]** of this document.

7.3 Report Conclusions

Once we have implemented algorithms in both tools while we monitored the evolution of the hardware we have reached the following conclusions.

Based on the experiments conducted and results obtained, the plant Spark architecture, is an evolution in data analysis tools. Overall All algorithms executed in Flink have run faster and with fewer complications.



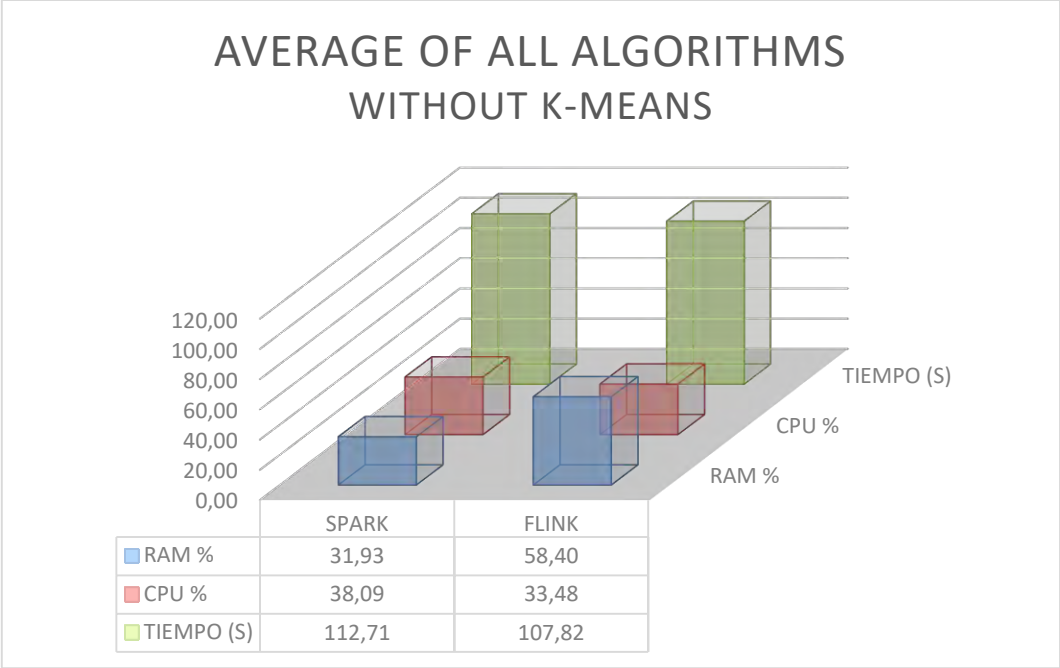
	SPARK	FLINK
RAM %	34,23	59,54
CPU %	39,23	38,04
TIEMPO (S)	267,55	83,47

The chart above shows us a global perspective, it is true that the average is not the best statistical estimate, we consider that this scenario we can use it in different experiments not extreme results that can undermine that conclusion is there.

As we have seen throughout all tests, Spark takes up more memory, around 20% more than Spark.

Regarding the use of CPU, the differences are not significant, so you could say that both have a very similar performance in CPU usage.

And the time to complete analyzing algorithms, Flink complete tasks in less time. It is true that the biggest difference of time is given by K-Means, but if we analyze the results if you take into account the latter algorithm.



	SPARK	FLINK
RAM %	31,93	58,40
CPU %	38,09	33,48
TIEMPO (S)	112,71	107,82

It is noted that the values of CPU / RAM remain constant with respect to the results with all algorithms. Where there really is a change, it is in the average response time which happens to be a difference of 184 seconds at just 5 seconds, but if we analyze one by one the cases we see a trend of improvement in progression and Flink.

So we conclude that Flink is a tool that implements a tangible improvement in the information processing tools or tools Big Data.

7.4 Futures Lines

We'll detail some improvements that could be introduced to the project developed:

- Docker prepare images for the Benchmark running on a real cluster with a large number of computers.
- Add an interface that allows you to compare tests that have launched individually, with the same interface as the comparative tests are shown.
- Allow loading of evaluation data from internet dynamically (at the time of the test launch proceeds to download the data).
- Add a test mode 3 Frameworks running simultaneously in virtualized environments and independent.
- Broadening the algorithms used in the evaluation of the tols.

Bibliografía

[1] winshuttle, «winshuttle,» big data historia cronologica, [En línea]. Available: http://www.winshuttle.es/big-data-historia-cronologica/. [Último acceso: 23 07 2016].

[2] S. Sánchez, «ctisoluciones,» Big Data: El camino desde el conocimiento hasta la creación de valor, [En línea]. Available: http://www.ctisoluciones.com/big-data-conocimiento-y-creacion-de-valor/. [Último acceso: 11 07 2016].

[3] GABRIEL, «ABC,» Cómo configurar WhatsApp para evitar compartir datos con Facebook, [En línea]. Available: http://www.abc.es/tecnologia/moviles/aplicaciones/abci-como-configurar-whatsapp-para-evitar-compartir-datos-facebook-201608291648_noticia.html. [Último acceso: 18 08 2016].

[4] M. Ward, «BBC,» Cómo se puede vaticinar el crimen usando "Big Data", 01 20 2014. [En línea]. Available: http://www.bbc.com/mundo/noticias/2014/04/140412_tecnologia_combate_crimen_wbm_finde. [Último acceso: 13 08 2016].

[5] N. -. L. Juridicas, «Noticias Juridicas,» Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal., [En línea]. Available: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.t2.html. [Último acceso: 5 09 2016].

[6] N. J. -. LPI, «Noticias Jurídicas,» Ley de Propiedad Intelectual, [En línea]. Available: http://noticias.juridicas.com/base_datos/Admin/rdleg1-1996.html. [Último acceso: 10 09 2016].

[7] yahoo!, «github,» Yahoo! Streaming Berchmarks, [En línea]. Available: https://github.com/yahoo/streaming-benchmarks/.

[8] Intel, «github,» Intel Benchmark Apache Storm, [En línea]. Available: https://github.com/intel-hadoop/storm-benchmark. [Último acceso: 17 07 2016].

[9] A. Flink, «Apache Flink,» Apache Flink Configuración General, [En línea]. Available: https://ci.apache.org/projects/flink/flink-docs-release-1.1/setup/config.html. [Último acceso: 10 07 2016].

[10] A. Hadoop, «Apache Hadoop,» Hadoop Software , [En línea]. Available: http://hadoop.apache.org/. [Último acceso: 6 08 2016].

[11] J. D. a. S. -. M. Ghemawat, «Google,» 12 2004. [En línea]. Available: https://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf. [Último acceso: 01 09 2016].

[12] A. Spark, «Apache Spark,» Apche Spark Configuración de la herramienta, [En línea]. Available: http://spark.apache.org/docs/latest/configuration.html. [Último acceso: 01 07 2016].

[13] S. P. Guide, «Spark,» Resilient Distributed Datasets (rdd), [En línea]. Available: http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds.

[14] tutorialspoint, «https://www.tutorialspoint.com,» Apache Storm Concepts, [En línea]. Available: https://www.tutorialspoint.com/apache_storm/apache_storm_core_concepts.htm. [Último acceso: 22 08 2016].

[15] t. -. Storm, «https://www.tutorialspoint.com,» Apache Storm Architecture, [En línea]. Available: https://www.tutorialspoint.com/apache_storm/apache_storm_cluster_architecture.htm. [Último acceso: 22 08 2016].

[16] Symfony, «Symfony,» Symfony Software, [En línea]. Available: https://symfony.com/. [Último acceso: 10 08 2016].

[17] Docker, «Docker,» Docker Guia de utilización, [En línea]. Available: https://docs.docker.com/engine/userguide/intro/. [Último acceso: 3 07 2016].

[18] A. Storm, «Apache Storm,» Apache Storm - Documentación Oficial, [En línea]. Available: http://storm.apache.org/releases/1.0.1/index.html. [Último acceso: 05 07 2016].

[19] I. Rogers, «Princeton University,» Princeton University - Page Rank Explained, [En línea]. Available: http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm. [Último acceso: 05 09 2016].

[20] C. Piech, «http://stanford.edu/,» K Means - Explicación del algoritmo y su funcionamiento, 2013. [En línea]. Available:

http://stanford.edu/~cpiech/cs221/handouts/kmeans.html. [Último acceso: 15 09 2016].

[21] A. Flink, «Flink DataSet,» Apche Flink - Documentación DataSet, [En línea]. Available: https://ci.apache.org/projects/flink/flink-docs-master/dev/batch/index.html. [Último acceso: 14 08 2016].

[22] Docker, «Docker,» Docker - Información para la instalación en entornos unix, [En línea]. Available: https://docs.docker.com/engine/installation/linux/ubuntulinux/. [Último acceso: 1 06 2016].

Anexo I: Resultados

Se adjuntan los gráficos con los diversos resultados obtenidos sobre los algoritmos analizados.

WordCount

ALGORITMO: WORD COUNT												
TEST	CPU	RAM (GB)	F	SPARK		TIEMPO (S)		FLINK		RESULTADO		EVOLUCIÓN TIEMPO (S)
				% RAM MEDIA	% CPU MEDIA	SPARK	FLINK	% RAM MEDIA	% CPU MEDIA	S	F	
1	1	1	1	28,71	21,4	143	116	39,06	21,99	ok	ok	SPARK
2			2	28,63	17,2	118	169	40,36	18,59	ok	ok	233,55
3			3	38,72	22,3	459	493	49,43	21,47	ok	ok	
4		2	1	48,4	21,4	141	150	69,75	20,98	ok	ok	
5			2	48,43	17,7	115	185	70,09	17,01	ok	ok	
6			3	49,94	17,9	434	564	67,74	18,2	ok	ok	FLINK
7		4	1	54,23	19,1	140	169	90,55	20,14	ok	ok	289,44
8			2	31,98	16,6	114	177	89,77	18,23	ok	ok	
9			3	38,48	17,3	438	582	88,77	18,88	ok	ok	
10	2	1	1	23,96	34,3	119	100	34,32	33,11	ok	ok	SPARK
11			2	23,53	30,6	70	95	34,72	29,9	ok	ok	142,78
12			3	25,95	34,3	292	299	36,38	33,74	ok	ok	
13		2	1	31,11	33,4	91	82	51,81	31,74	ok	ok	
14			2	29,49	29	68	96	53,71	30,78	ok	ok	
15			3	35,48	33,3	255	320	87,97	30,65	ok	ok	FLINK
16		4	1	42,25	30,4	82	92	86,11	31,26	ok	ok	165,33
17			2	36,52	28,4	66	96	86,77	30,86	ok	ok	
18			3	48,26	31,1	242	308	88,04	30,51	ok	ok	
19	4	1	1	23,97	50,2	101	77	34,96	45,24	ok	ok	SPARK
20			2	23,96	48,6	50	66	35,44	46,77	ok	ok	104,33
21			3	26,74	54,8	241	176	52,94	58,78	ok	ok	
22		2	1	32,15	50,2	69	48	51,99	50,62	ok	ok	
23			2	30,28	47,8	46	56	52,76	48,65	ok	ok	
24			3	35,26	54,1	171	176	54,96	58,35	ok	ok	FLINK
25		4	1	44,63	49,6	62	55	86,87	48,61	ok	ok	98,55
26			2	37,49	47,2	44	59	85,49	48,91	ok	ok	
27			3	49,4	53,9	155	174	87,62	56,44	ok	ok	
28	8	1	1	25,01	70,9	94	49	34,92	65,39	ok	ok	SPARK

29			2	23,57	67,5	48	51	34,61	73,43	ok	ok	
30			3	26,24	78,6	244	140	53,32	83,2	ok	ok	101,66
31			1	34,11	71,3	78	47	53,7	67,81	ok	ok	
32		2	2	31,49	67,5	40	49	54,61	75,47	ok	ok	
33			3	36,51	78,6	177	144	56,2	82,82	ok	ok	FLINK
34			1	44,68	63,9	60	56	85,14	67,18	ok	ok	
35		4	2	35,43	69,7	38	49	83,06	79,45	ok	ok	80,22
36			3	45,9	83,4	136	137	85,48	85,43	ok	ok	

Tabla XXXIV WordCount: Tabla de resultados

PageRank

ALGORITMO: PAGERANK												
TEST	CPU	RAM (GB)	F	SPARK		TIEMPO (S)		FLINK		RESULTADO		EVOLUCIÓN TIEMPO (S)
				% RAM MEDIA	% CPU MEDIA	SPARK	FLINK	% RAM MEDIA	% CPU MEDIA	S	F	
1	1	1	1	20,68	24,07	60	19	29,15	23,83	ok	ok	SPARK
2			2	21,92	22,61	119	43	31,97	19,82	ok	ok	88,77
3			3	22,61	23,04	238	103	33,17	17,12	ok	ok	
4		2	1	21,68	22,49	30	19	31,28	21,33	ok	ok	
5			2	23,66	26,15	55	36	46,24	22,55	ok	ok	
6			3	26,34	26,37	115	94	48,01	21,28	ok	ok	FLINK
7		4	1	29,13	29,11	27	20	49,51	19,63	ok	ok	50,77
8			2	25,13	22,51	51	40	75,74	27,71	ok	ok	
9			3	33,83	23,41	104	83	80,09	23,93	ok	ok	
10	2	1	1	41,42	23,33	59	22	83,81	20,21	ok	ok	SPARK
11			2	22,62	33,8	86	47	32,93	20,77	ok	ok	79,44
12			3	30,68	41,49	189	104	33,47	21,83	ok	ok	
13		2	1	24,51	26,86	29	19	46,31	24,73	ok	ok	
14			2	29,31	31,11	72	42	49,71	19,68	ok	ok	
15			3	32,14	35,55	144	103	52,52	17,9	ok	ok	FLINK
16		4	1	26,64	23,58	27	20	77,14	29,02	ok	ok	54,55
17			2	34,56	33,12	38	40	81,39	23,79	ok	ok	
18			3	42,33	36,83	71	94	85,96	19,46	ok	ok	
19	4	1	1	22,44	22,4	60	25	32,38	21,11	ok	ok	SPARK
20			2	22,99	36,74	92	45	33,44	22,21	ok	ok	79,44
21			3	24,31	64,52	182	127	34,62	19,86	ok	ok	
22		2	1	25,14	25,1	29	22	48,25	22,5	ok	ok	
23			2	30,58	30,94	73	45	50,55	21,35	ok	ok	
24			3	32,78	59,54	137	103	52,09	20,92	ok	ok	FLINK
25		4	1	27,02	22,19	27	20	77,68	27,53	ok	ok	60,33
26			2	36,36	34,54	37	48	82,43	24,34	ok	ok	
27			3	43,3	46,95	78	108	85,93	19,38	ok	ok	
28	8	1	1	21,13	23,63	60	23	30,48	23,84	ok	ok	SPARK
29			2	22,89	34,82	90	47	32,77	24,57	ok	ok	71,66
30			3	23,6	51,8	136	118	33	23,35	ok	ok	
31		2	1	23,81	25,26	29	24	47,93	24,14	ok	ok	
32			2	28,77	32,91	60	45	50,07	24,12	ok	ok	
33			3	31,87	57,93	126	129	53,01	20,34	ok	ok	FLINK

34		4	1	26,14	23,42	27	24	77,71	28,6	ok	ok	63,33
35			2	33,68	34,23	38	48	81,1	24,72	ok	ok	
36			3	41,88	47,74	79	112	83,82	22,35	ok	ok	

Tabla XXXV PageRank: Tabla de resultados

K-Means

ALGORITMO: K-MEANS												
TEST	CPU	RAM (GB)	F	SPARK		TIEMPO (S)		FLINK		RESULTADO		EVOLUCIÓN TIEMPO (S)
				% RAM MEDIA	% CPU MEDIA	SPARK	FLINK	% RAM MEDIA	% CPU MEDIA	S	F	
1	1	1	1	31,44	24,98	107	7	40,16	27,61	ok	ok	SPARK
2			2	37,86	59,81	931	30	52,58	53,14	ok	ok	717,66
3			3	64,14	9,66	2000	131	59,06	76,55	E01	ok	
4		2	1	33,09	24,43	105	7	53,13	30,47	ok	ok	
5			2	40,65	25,3	386	20	67,68	40,89	ok	ok	
6			3	54,48	56,82	1691	53	72,58	58,58	ok	ok	FLINK
7		4	1	50,16	24,82	101	13	83,95	47,04	ok	ok	35,11
8			2	41,12	24,37	386	19	82,19	43,97	ok	ok	
9			3	46,3	23,57	752	36	87,05	47,93	ok	ok	
10	2	1	1	25,25	35,07	67	6	35,64	31,42	ok	ok	SPARK
11			2	32,93	65,36	736	30	41,39	53,64	ok	ok	551,88
12			3	54,48	56,82	1690	125	44,69	76,25	ok	ok	
13		2	1	32,03	33,56	69	6	55,05	31,13	ok	ok	
14			2	41,27	32,1	250	20	62,55	41,81	ok	ok	
15			3	45,53	67,49	1378	53	63,31	58,67	ok	ok	FLINK
16		4	1	41,48	31,95	68	10	80,85	37,95	ok	ok	34
17			2	41,68	31,97	242	20	81,24	42,21	ok	ok	
18			3	42,95	31,51	467	36	82,59	47,89	ok	ok	
19	4	1	1	21,76	42,13	52	6	32,34	28,91	ok	ok	SPARK
20			2	28,25	71,11	600	30	36,2	54,11	ok	ok	522,33
21			3	28,58	5,09	2000	129	37,02	76,77	E01	ok	
22		2	1	24,34	42,7	51	6	45,8	30,86	ok	ok	
23			2	31,44	46,22	178	20	54,54	41,94	ok	ok	
24			3	45,27	75,86	1240	54	77,86	58,87	ok	ok	FLINK
25		4	1	60,8	42,37	50	13	86,07	41,91	ok	ok	34,88
26			2	38,06	46,14	174	18	79,59	42,11	ok	ok	
27			3	37,73	43,12	356	38	82,23	47,57	ok	ok	
28	8	1	1	21,23	42,09	54	6	31,56	28,32	ok	ok	SPARK
29			2	27,14	73,86	659	29	35,04	52,61	ok	ok	517,11
30			3	30,66	5,16	2000	129	48,04	76,64	E01	ok	
31		2	1	35,31	42,17	52	6	56,93	29,81	Ok	ok	
32			2	40,89	46,74	177	20	62,3	41,71	Ok	ok	

33			**3**	47,82	75,49	**1145**	**53**	64,95	58,05	**Ok**	**ok**	FLINK
34		**4**	**1**	41,62	41,79	**52**	**17**	85,18	49,68	**Ok**	**ok**	**35,11**
35			**2**	40,85	45,65	**179**	**18**	81,32	42,58	**Ok**	**ok**	
36			**3**	40,28	47,21	**336**	**38**	82,55	48,59	**ok**	**ok**	

Tabla XXXVI K-Means: Tabla de Resultados

Anexo II: Manual de Instalación

En este anexo indicamos como se debe realizar el proceso de instalación y configuración de la aplicación para su uso.

Este manual de instalación se centrará en guiar el proceso de instalación en un entorno UNIX/Ubuntu, pero es similar para otras plataformas.

El primer paso que debemos realizar, es la instalación del comando git, para poder descargar de los repositorios, todos los paquetes necesarios, para ello ejecutamos el siguiente comando.

```
sudo apt-get install git
```

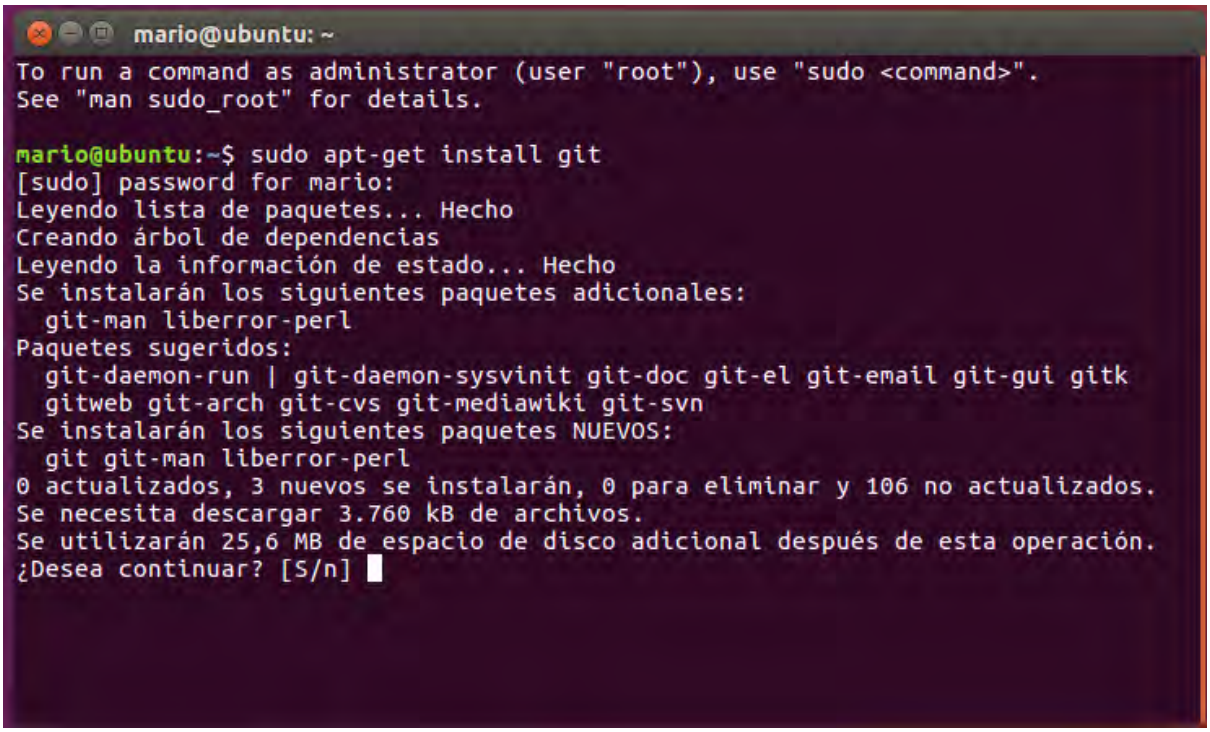


Ilustración XXIV Manual de Instalación

Como se observa en la imagen, pulsaremos “S” para indicar que se acepta la instalación de paquetes adicionales.

El siguiente paso que debemos realizar, es la descarga del paquete, que realizar la instalación de todo el entorno, ejecutamos:

```
git clone https://github.com/MarioSkill/TFG-install.git
```

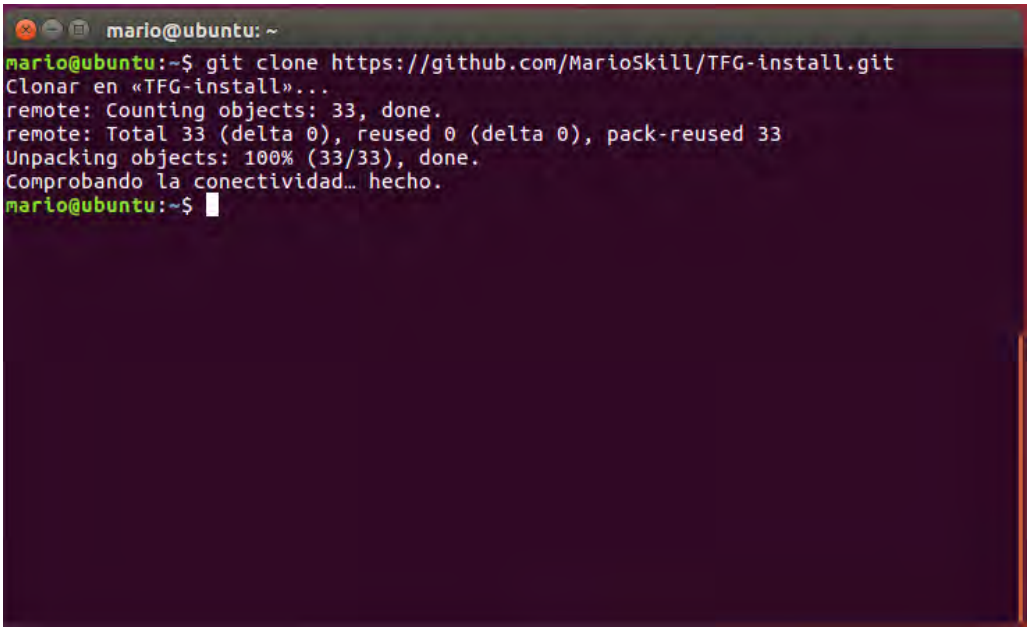


Ilustración XXV Manual de Instalación 2

En la imagen se observa, que se ha clonado el repositorio TFG-install a la ubicación en la que nos encontrábamos, en este caso en la carpeta personal, si ejecutamos "ls", podremos ver las carpetas que hay en el directorio y veremos que se ha creado un directorio con el mismo nombre que el repositorio que hemos clonado. El repositorio se encuentra alojado en GitHub y se puede acceder a través de la siguiente dirección https://github.com/MarioSkill/TFG-install.

El repositorio cuenta con:

I. Carpeta files: Archivos de configuración necesario para el funcionamiento de alguno de los componentes.
II. Script install.sh: script que permite la instalación automática de los componentes.
III. Script tfg.sql: script sql que permite crear la estructura de base de datos necesaria para almacenar las pruebas.

IV. Archivo Readme.md con información del repositorio.
V. Imagen de ejemplo del proceso de instalación.

El siguiente paso que debemos realizar es:

1. Entrar al directorio TFG-install.
2. Dar permisos de ejecución al script install.sh mediante, chmod 755.
3. ejecutar el script y seguir los pasos que aparecen en pantalla.

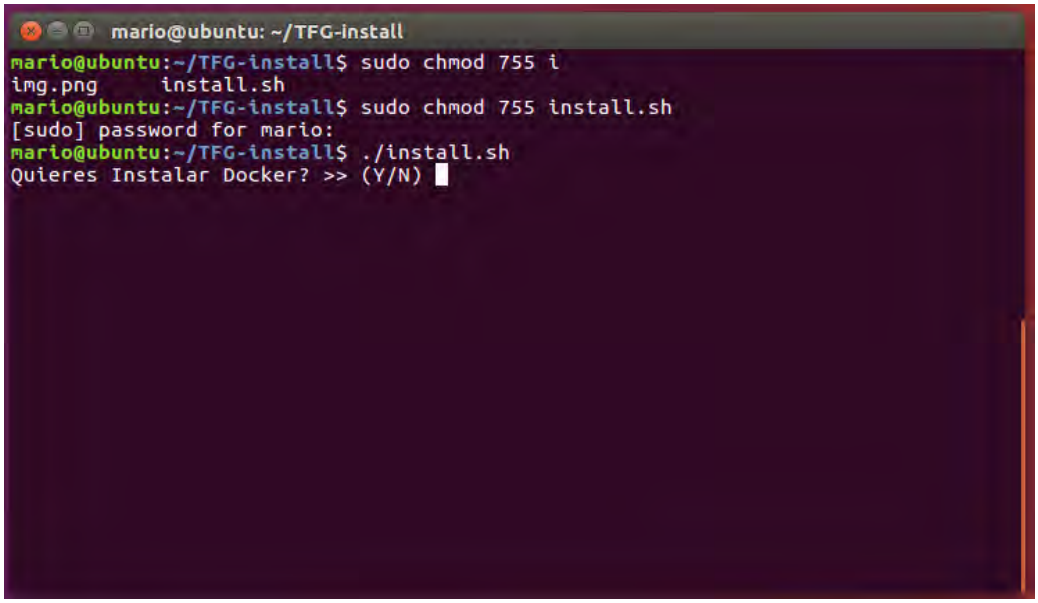


Ilustración XXVI Manual de Instalación: Componentes

Como se puede ver en la imagen, el script nos irá preguntando si queremos instalar el paquete que corresponda. El script está preparado para instalar los siguientes componentes:

- Docker
- Stack Lamp (con soporte multi-Hilo)
  - Php, MySql y Apache2
- phpMyAdmin
- Symfony 3
- Bundle TFG

Adicionalmente, importará la base de datos necesaria para el funcionamiento de este proyecto y descargará las imágenes previamente creadas de Docker con las herramientas que vamos a analizar:

- Apache Storm
- Apache Spark
- Apache Flink

Dichas imágenes se puede encontrar en https://hub.docker.com/r/marioskill o en https://github.com/MarioSkill.

El siguiente paso que debemos realizar es configurar el acceso a la BBDD desde Symfony, para esta tarea nos dirigimos a:

/var/www/default/TFG/app/config/parameters.yml

Esta ruta variará, dependiendo de donde tengamos configurado la ruta del servidor web, por defecto es: **/var/www/default/.** En este archivo deberemos indicar el nombre de BBDD, el usuario de acceso y contraseña para acceder a MySql.

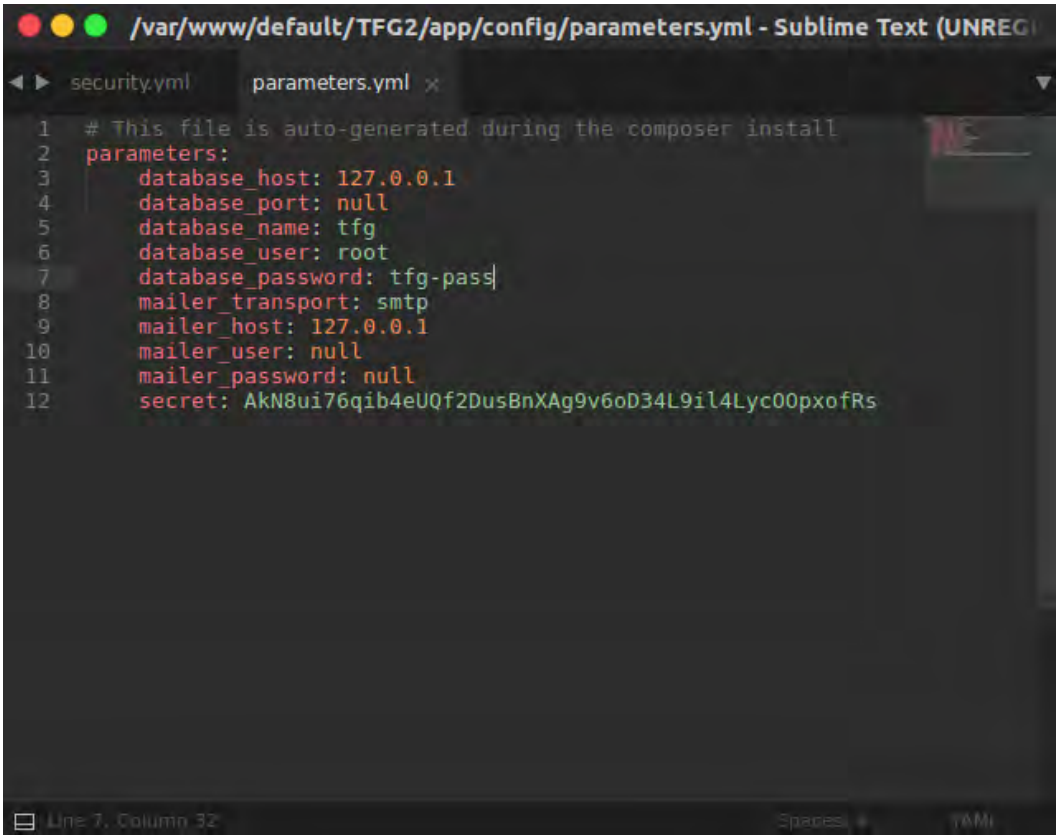


Ilustración XXVII Configuración Symfony

Otro de los archivos de configuración que debemos modificar es el encargado de la seguridad de la aplicación, y nos permite indicar el algoritmo de codificación de contraseña, qué páginas requieren que el usuario tenga una sesión iniciada.
El archivo con toda la configuración se puede copiar de la carpeta **TFG-install** (con la que iniciamos este manual) dentro de la carpeta files, copiamos el archivo security.yml a:

/var/www/default/TFG/app/config/security.yml

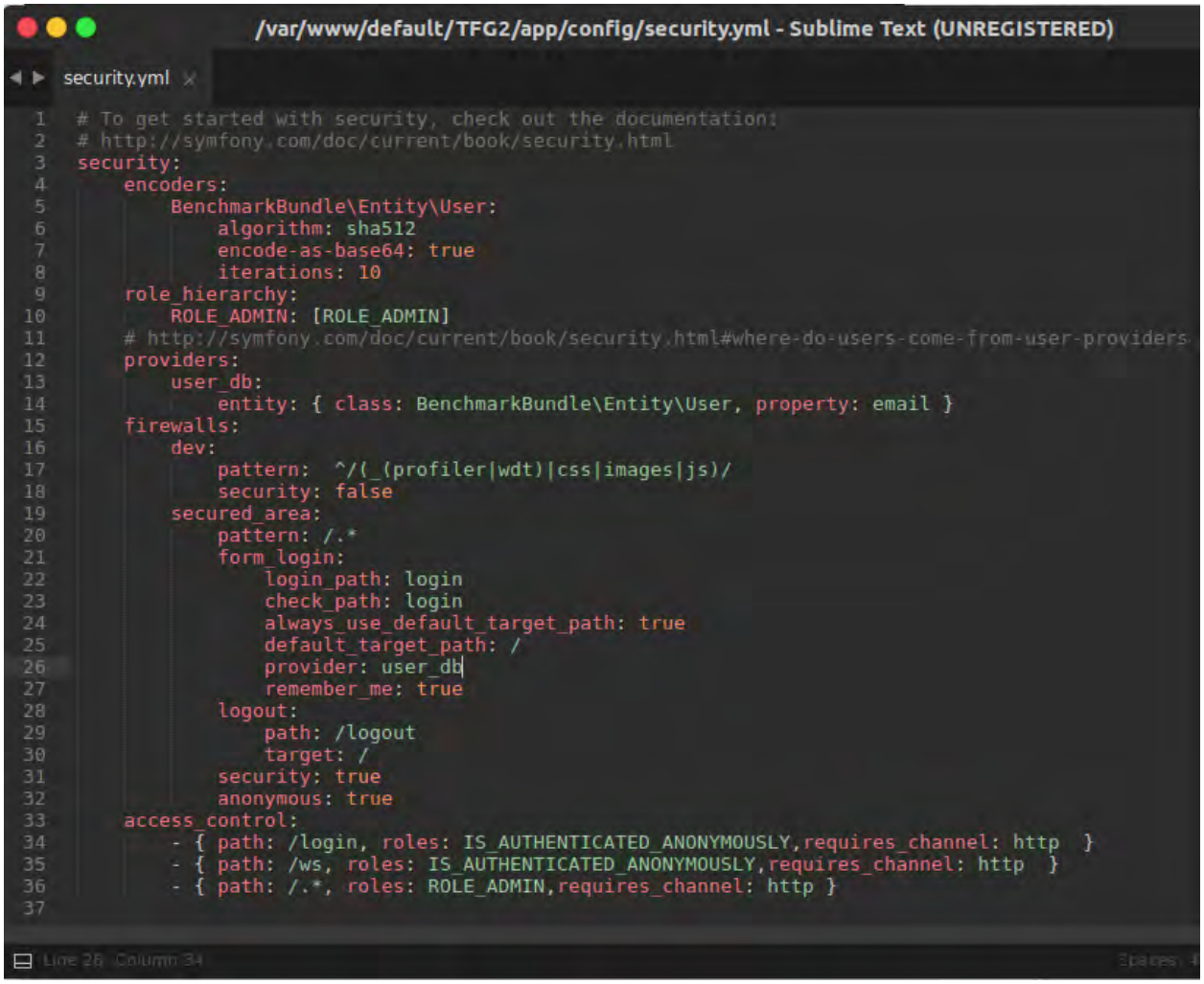


Ilustración XXVIII Configuración Symfony 2

Después de completar todos estos pasos si accedemos a la siguiente url:

http://localhost/TFG2/web/app_dev.php/

Nos aparecerá la ventana de login, en la cual podremos iniciar sesión con las credenciales:

- Correo Electronico: 100292688@alumnos.uc3m.es
- Contraseña: mario

Ilustración XXIX Login Aplicación

Si el inicio de sesión es satisfactorio, podremos acceder al dashboard o tablero de administración.

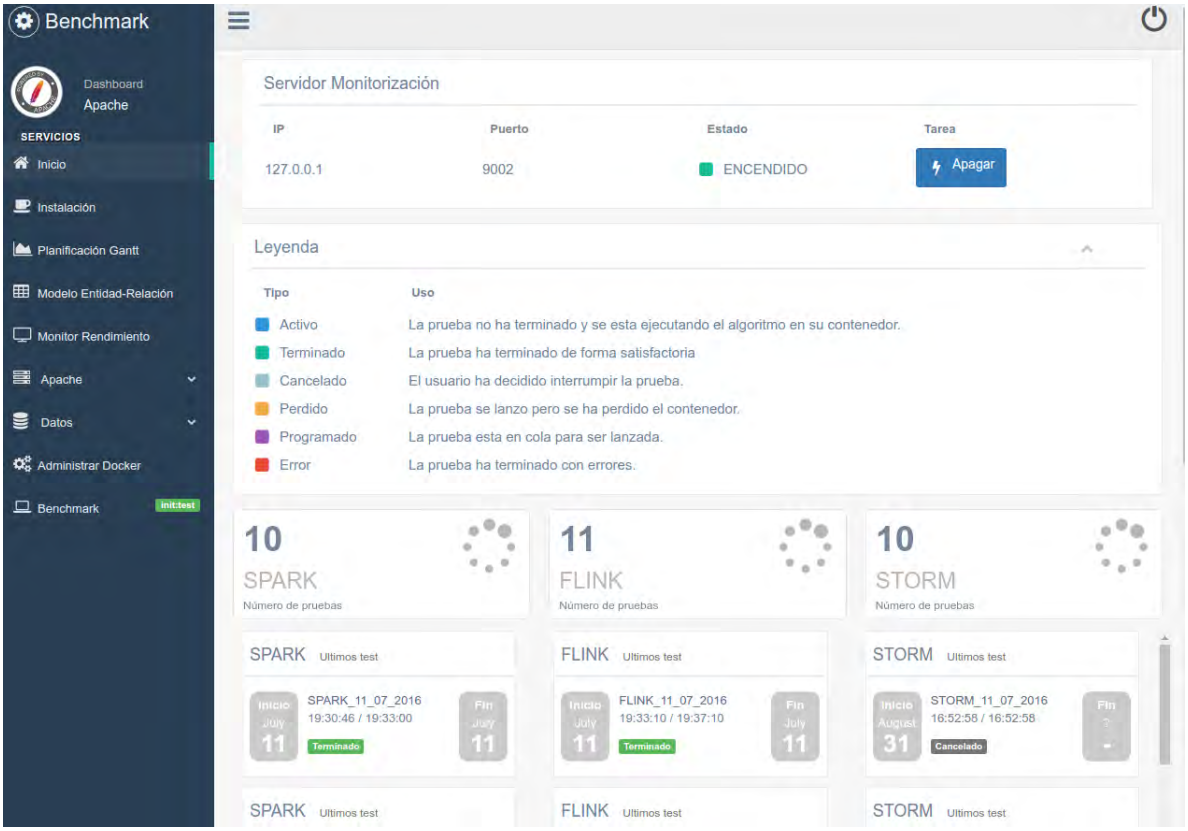


Ilustración XXX Aplicación Home

En este punto ya podremos empezar a lanzar pruebas y ver los resultados obtenidos de cada prueba individual o de la comparativa entre los tres Frameworks.

En la siguiente imagen se puede ver un ejemplo graficado de los resultados de analizar un fichero de texto con un algoritmo. La primera gráficas nos indican el tiempo en completar una tarea y la segunda gráfica podemos ver la evolución de uso de memoria RAM y de CPU a lo largo que dura la prueba.

Anexo III: Interfaz de la aplicación

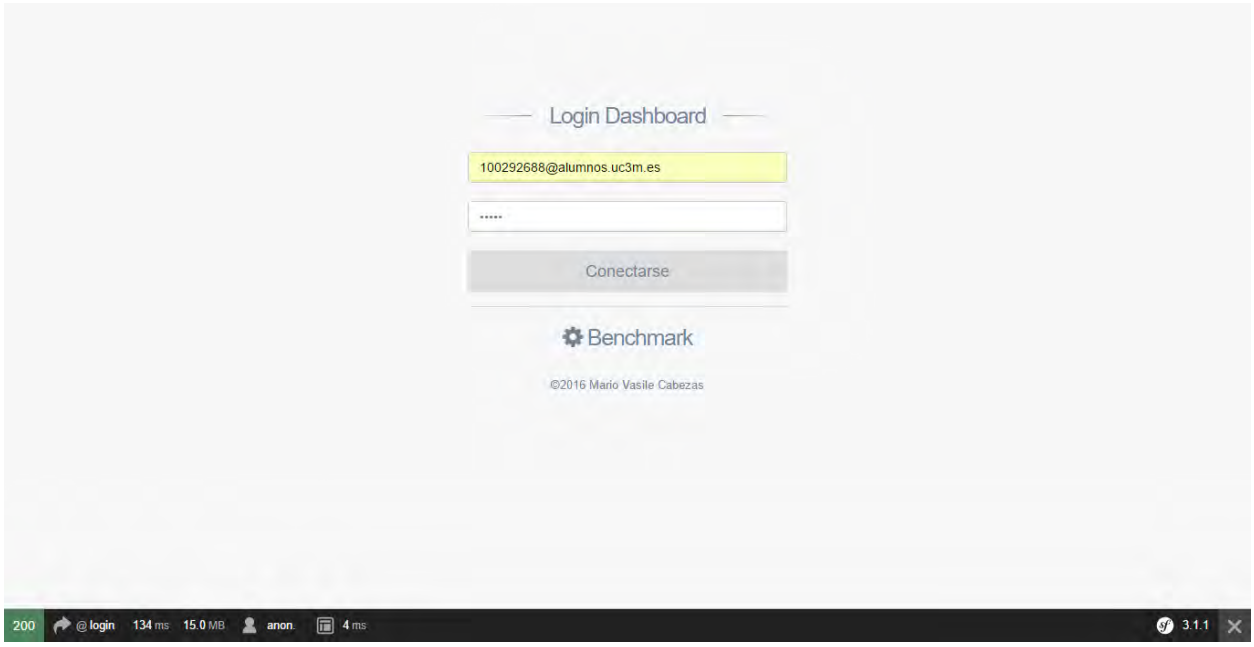


Ilustración XXXI Login Aplicación Final

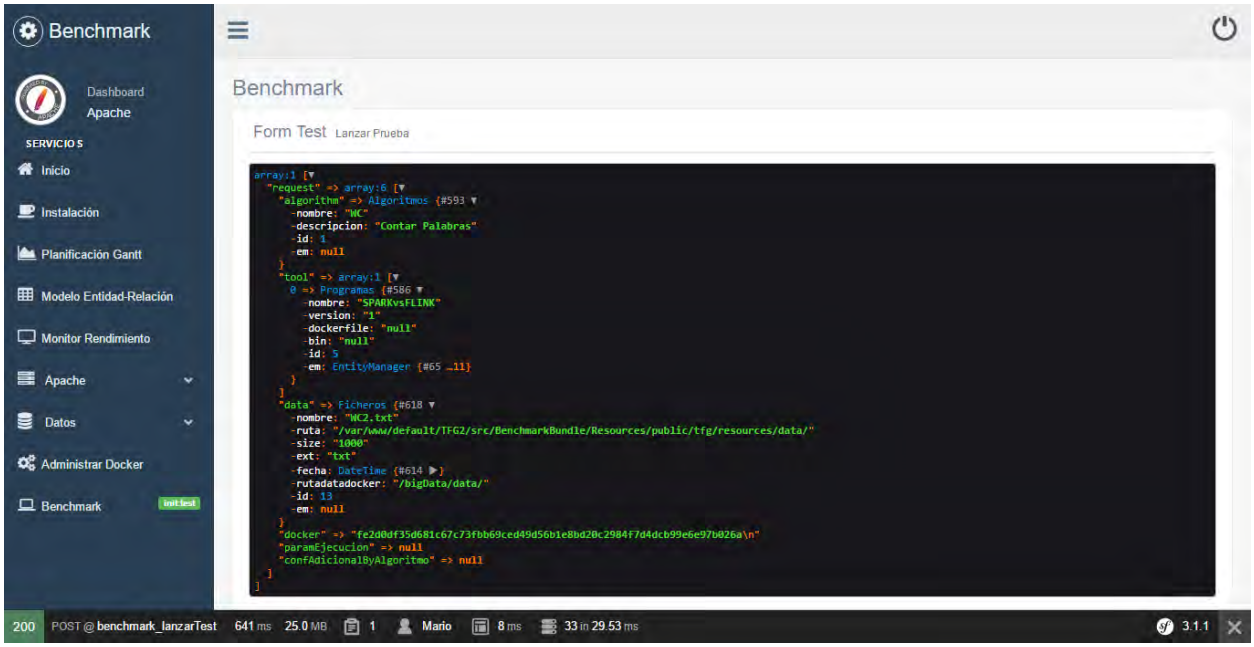


Ilustración XXXII Lanzar Prueba Aplicación

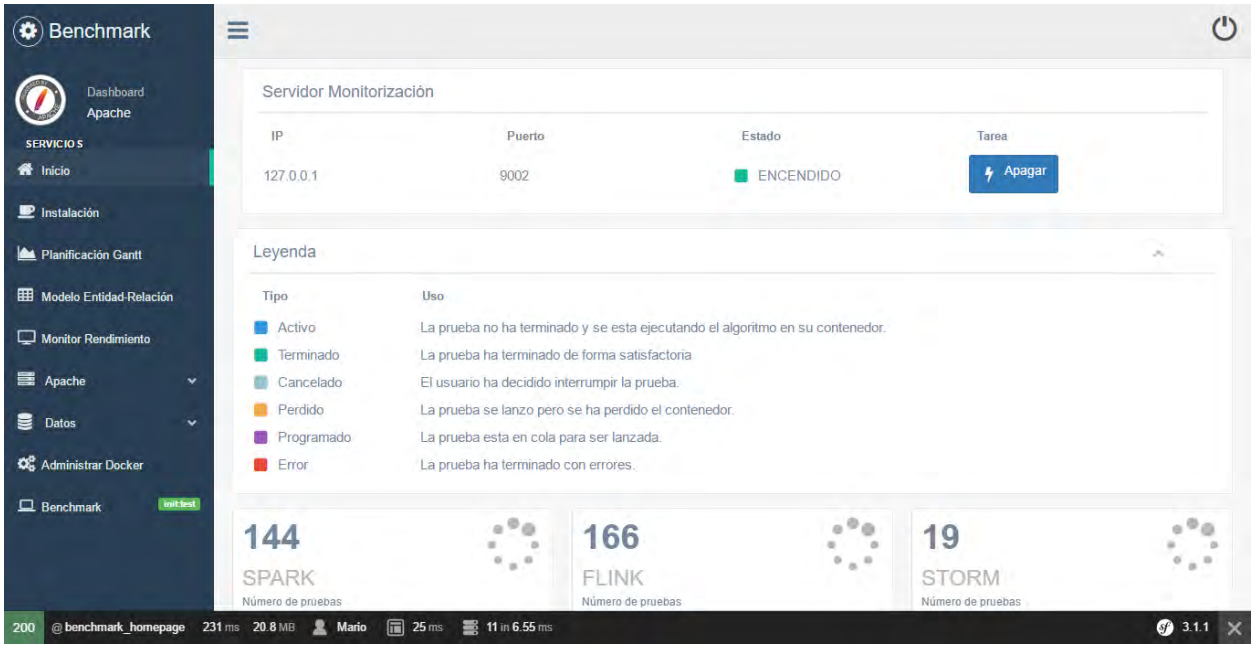


Ilustración XXXIII Página Principal de la Aplicación 1

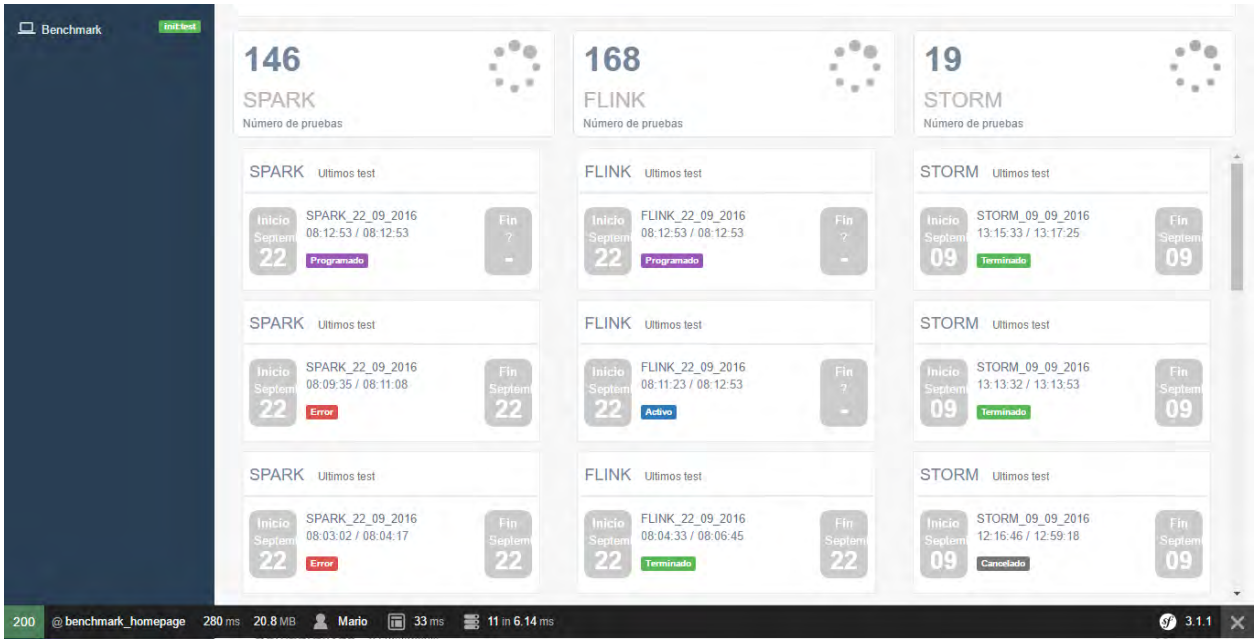


Ilustración XXXIV Página Principal de la Aplicación 2

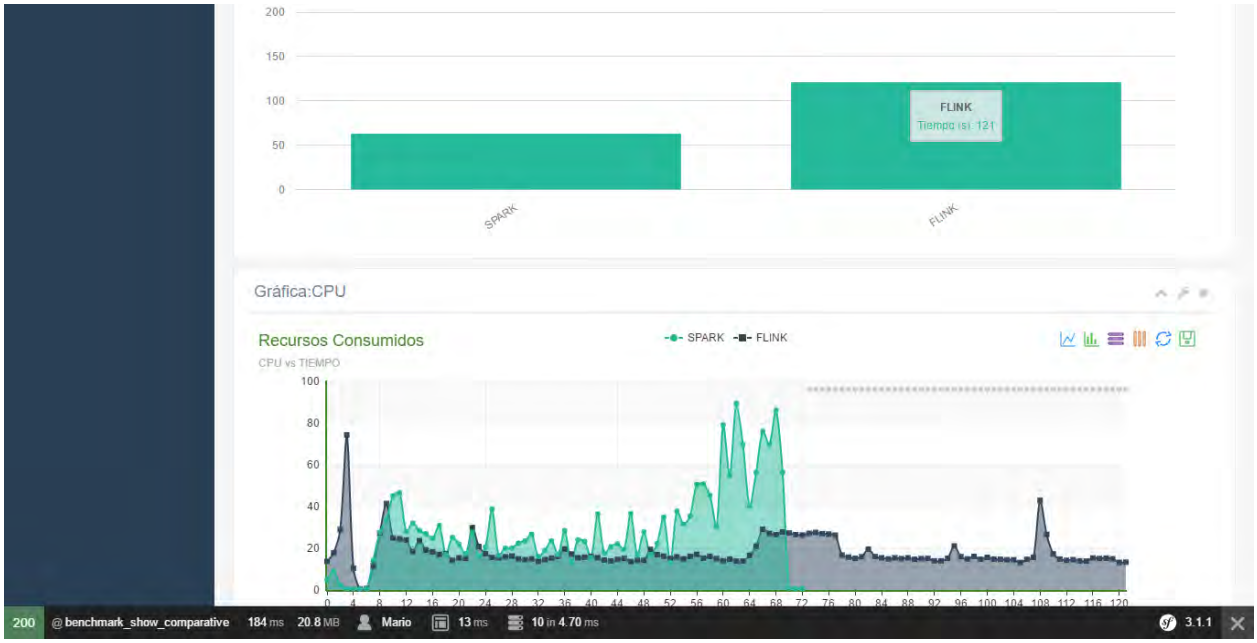


Ilustración XXXV Resultados de la ejecución

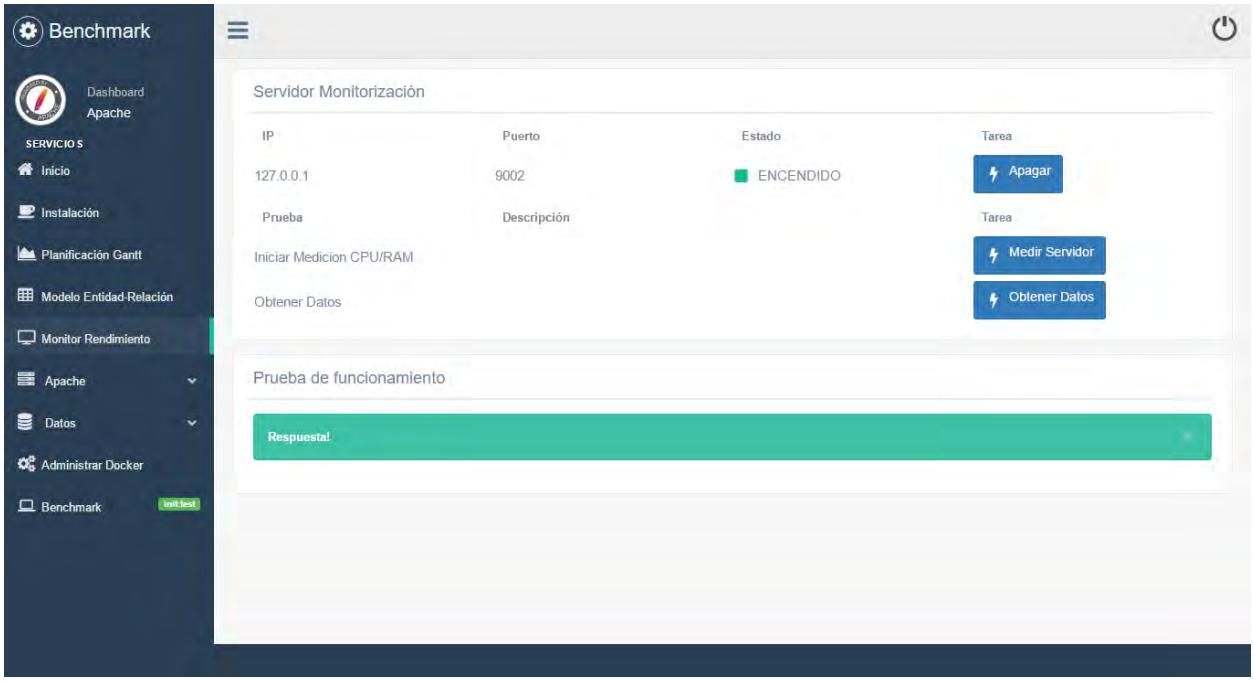


Ilustración XXXVI Panel de control del Monitor de rendimiento

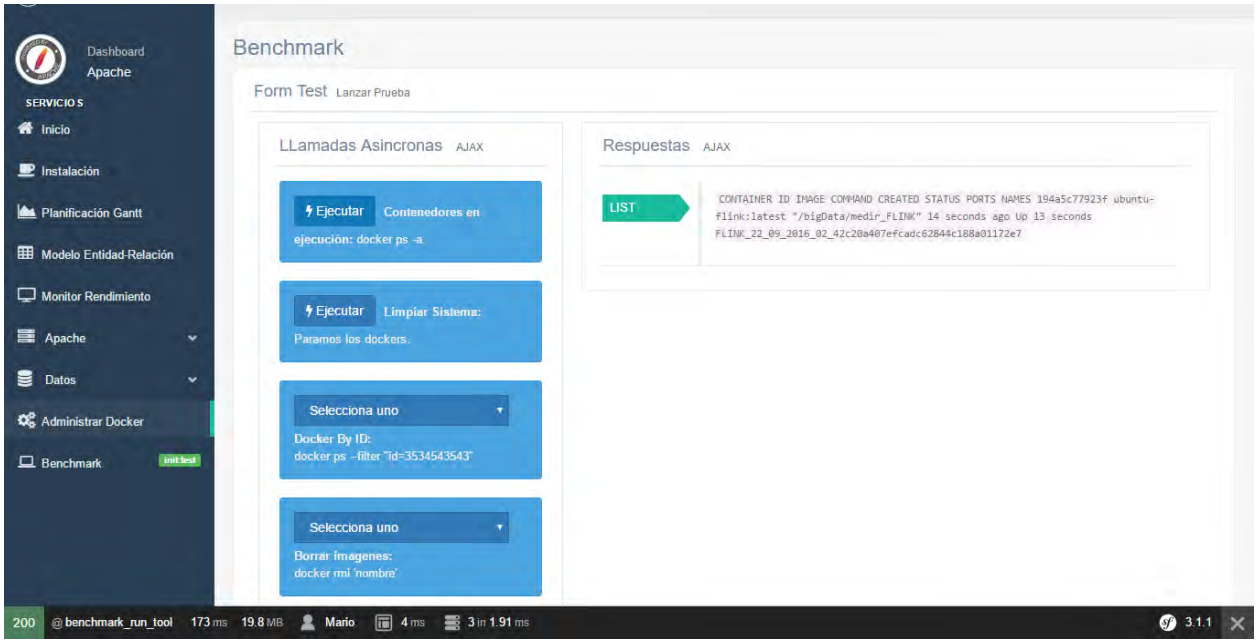


Ilustración XXXVII Panel de control de Docker