UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Sistema de procesado y análisis de Big Data en Smart Buildings

Autor:
Xosé Fernández Fuentes

Directores:
José Ángel Taboada González
David Mera Pérez

Grado en Ingeniería Informática

Julio 2016

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

D. José Ángel Taboada González, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D. David Mera Pérez**, Investigador postdoctoral de la Universidad de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *Sistema de procesado y análisis de Big Data en Smart Buildings*, presentada por **D. Xosé Fernández Fuentes** para superar los créditos correspondientes al Trabajo de Fin de Grao de la titulación del Grao en Ingeniería Informática, se realizó bajo nuestra dirección en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 8 de julio de 2016:

O director, O codirector, O alumno,

José Ángel Taboada González David Mera Pérez Xosé Fernández Fuentes

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este proyecto se propone el despliegue de un sistema distribuido que sea capaz de almacenar, procesar y analizar toda la información generada por los sensores de un edificio inteligente. Dicho sistema deberá atender y dar respuesta en tiempo real a las consultas generadas por los usuarios. Los resultados a las consultas mostrarán una imagen global y actual del sistema por lo que éstos deberán integrar tanto la información del histórico como la generada en tiempo real.

1.1. Contexto

Los recientes avances, tanto en hardware como en software, nos permiten digitalizar una gran parte del mundo que nos rodea y generar ingentes cantidades de datos. Estamos ante la era denominada del *Big Data*, donde cada vez es más común que las organizaciones tengan acceso a más datos de los que son capaces de procesar (Figura 1.1).

Un caso particular del *Big Data* lo encontramos en los edificios inteligentes (*Smart Buildings*), en los que están presentes amplias redes de sensores heterogéneos que nos permiten monitorizar cada aspecto del edificio. La información recibida a través de dichas redes permite gestionar los sistemas de los edificios, como por ejemplo, adaptar la iluminación dependiendo de la luz exterior y la hora del día, regular el aire acondicionado dependiendo de la época del año y del número de personas en una habitación, detectar la necesidad de mantenimien-

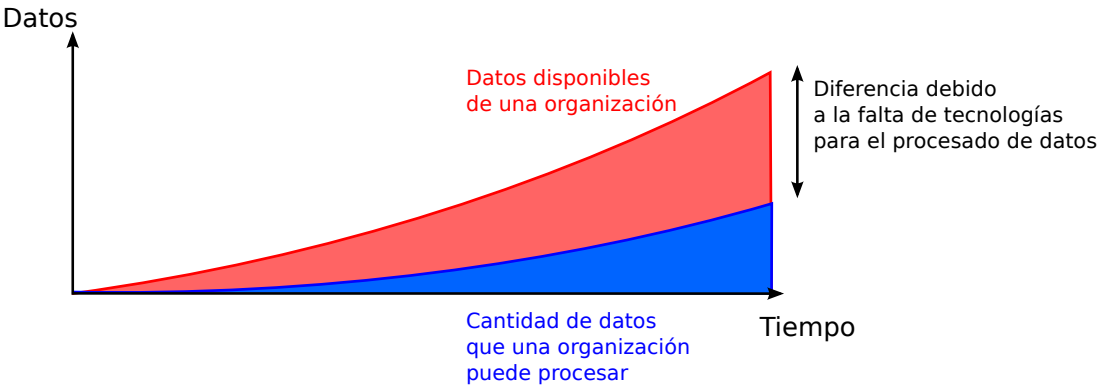


Figura 1.1: Datos disponibles a una organización vs. Datos que puede procesar una organización

to (como la sustitución de una bombilla o sensor averiado), controlar el suelo radiante, monitorizar células fotovoltaicas, etc.

Los edificios inteligentes generan grandes volúmenes de datos que son a menudo infrautilizados debido a que suelen contener información y relaciones no evidentes que requieren de un procesado exhaustivo y su posterior integración. Un ejemplo de *Smart Buildings* se encuentra en la Universidad de Santiago de Compostela, donde se han desplegado redes de sensores en 54 de sus edificios. Estas redes pueden llegar a generar más de 10.000 señales cada segundo que deben de ser almacenadas, procesadas y analizadas. En la actualidad cada edificio almacena sus propios datos en una base de datos relacional clásica que es consultada por diferentes procesos para extraer información que, posteriormente, será utilizada para administrar los edificios. La solución actual para la gestión de los datos empieza a mostrar problemas de escalado, ya que a medida que los edificios y los sensores aumentan, la calidad del sistema se degrada. Diferentes optimizaciones como, por ejemplo, mejoras en el modelo relacional de la BD, muestreo de las señales cada 10 segundos o el análisis de datos agrupados cada diez minutos, se han ido llevando a cabo para que el sistema pueda gestionar el volumen de datos actual. Sin embargo, el crecimiento exponencial de los datos requiere de un nuevo enfoque pensando en el futuro a corto y medio plazo. El análisis y procesamiento de grandes volúmenes de datos históricos ya es de por si una tarea ardua pero, además, para una correcta interpretación del momento actual, es necesario integrar los resultados de dicho procesado con los datos en tiempo real. Los pa-

radigmas y tecnologías tradicionales muestran muchas limitaciones en la gestión y procesado de grandes volúmenes de datos. La utilización de nuevos paradigmas de computación distribuida parece ser la clave para enfrentarnos a los problemas relacionados con el *Big Data*.

1.2. Objetivos

Se propone desplegar un sistema distribuido para el almacenamiento, procesado y análisis de grandes volúmenes de datos proporcionados por la red de sensores desplegada en el edificio *CiTIUS*. Este sistema deberá ser capaz de:

- **OBJ-1**: Gestionar en tiempo real grandes volúmenes de datos procedentes de sensores.
- **OBJ-2**: Gestionar el gran volumen de datos históricos almacenado en el sistema.
- **OBJ-3**: Responder, en tiempo real, a consultas prediseñadas y parametrizables empleando todos los datos disponibles en el sistema hasta el momento de dicha consulta.
- **OBJ-4**: Garantizar el almacenamiento de los datos generados por el sistema en su forma original y de modo inmutable y permanente para que dichos datos puedan ser empleados en sistemas y/o consultas futuras no contempladas en la actualidad.

1.3. Organización del documento

Esta memoria está organizada en 9 capítulos y 2 apéndices. A continuación se muestra una pequeña descripción del contenido de cada uno de ellos:

- En el ***capítulo 1*** se expone el contexto y se platean los objetivos del proyecto.
- En el ***capítulo 2*** se describe la gestión realizada durante el trascurso del proyecto, comprendiendo el enunciado del alcance, la gestión de riesgos, la

gestión de la configuración, la metodología aplicada, la planificación y el análisis de costes.

- En el ***capítulo 3*** se muestra el análisis de requisitos realizado.
- En el ***capítulo 4*** se describe y explica el funcionamiento de cada una de las tecnologías utilizadas.
- En el ***capítulo 5*** se muestra el diseño realizado a alto nivel de cada uno de los subsistemas.
- En el ***capítulo 6*** se detalla cada uno de los subsistemas de la arquitectura desarrollada a bajo nivel.
- En el ***capítulo 7*** se expone como se desplegó cada una de las tecnologías en un cluster.
- En el ***capítulo 8*** se detalla el plan, el diseño y la ejecución de las pruebas realizadas.
- En el ***capítulo 9*** se recogen las conclusiones derivadas de la realización del proyecto así como las posibles ampliaciones.
- En el ***apéndice A*** se explica como instalar y desplegar toda la infraestructura utilizada.
- En el ***apéndice B*** se proporciona un pequeño manual de usuario.

Capítulo 2

Gestión del proyecto

En este capítulo se abordan todos los aspectos relacionados con la gestión del proyecto realizados a lo largo de desarrollo. Esta gestión está formada por: (i) el alcance, (ii) el análisis y gestión de riesgos, (iii) la gestión de la configuración, (iv) la descripción de la metodología de desarrollo, (v) la planificación temporal y (vi) el análisis de costes.

2.1. Enunciado del Alcance

En este apartado se detallan los siguientes puntos:

- **Descripción del alcance.** Se describen las características del producto esperado tras la realización del proyecto.
- **Criterios de aceptación del producto**. Se definen los criterios para aceptar el producto final.
- **Productos entregables del proyecto**. Se definen qué productos se entregan al cliente una vez finalizado el proyecto.
- **Exclusiones del proyecto**. Se describen qué características no son contempladas en el producto final.
- **Restricciones del proyecto**. Se enumeran las restricciones específicas del proyecto asociadas con el alcance del proyecto.

- **Supuestos del proyecto**. Se enumeran y describen las asunciones específicas del proyecto asociadas con el alcance del proyecto.

En el apartado del PMBOK [1] donde se describen los puntos que debe cubrir el Enunciado del Alcance hay más puntos de los que se contemplan aquí, como por ejemplo limitación de fondos o organización inicial del proyecto. Esos han sido excluidos debido a que no proceden en este proyecto.

Este punto en concreto, Enunciado del Alcance, se complementa con el Capítulo 3 (Análisis de requisitos), ya que en él se proporciona con un nivel de detalle mayor qué es lo que se aborda en este proyecto.

2.1.1. Descripción del alcance

El sistema a desarrollar será distribuido y deberá ser capaz de almacenar, procesar y analizar grandes volúmenes de datos. El sistema trabajará con datos procedentes de redes de sensores, en particular, utilizará los adquiridos en tiempo real por la red de sensores del *CiTIUS*. Una vez recogidos, deberá almacenarlos de forma permanente e inmutable en un sistema de ficheros distribuido, escalable y tolerante a errores. Después, deberá ser capaz de procesar todos los datos, unificando tanto los datos actuales como los históricos. De este procesado deberá generar unas vistas que serán almacenadas en una base de datos distribuida. Además, el sistema deberá disponer de un servicio a través del cual los usuarios puedan lanzar consultas. El resultado de dichas consultas provendrá de las vistas generadas previamente.

2.1.2. Criterios de aceptación del producto

En base a los objetivos establecidos para el proyecto, se definen los siguientes criterios de aceptación:

- El sistema da soporte al almacenamiento de los datos históricos generados por la red de sensores del *CiTIUS*. Estos datos ocupan actualmente 96GB y abarcan desde el 14 de enero de 2015 hasta el 1 de julio de 2016.

- El sistema es capaz de obtener y procesar en tiempo real el volumen de datos

generados por la red de sensores del edificio, el cual genera 667 señales cada 10 segundos.

- Los datos generados son almacenados en su forma original, de forma permanente e inmutable.
- El sistema es capaz de responder en tiempo real a consultas prediseñadas sobre el conjunto de datos (históricos y actuales) obtenidos a través de los sensores de consumo de agua y gas.
- El sistema es capaz de responder en tiempo real a consultas prediseñadas sobre el conjunto de datos (históricos y actuales) obtenidos a través de los sensores de temperatura instalados en los despachos del edificio.

2.1.3. Productos entregables del proyecto

Los productos entregables del proyecto tras su finalización son:

- La memoria del proyecto que incluya: todos los aspectos de gestión del proyecto, análisis, diseño e implementación del sistema, así como un manual de usuario y un manual técnico, que deberá contener todos los pasos necesarios para desplegar el sistema.
- Todo el software necesario para poner en marcha toda la infraestructura.

2.1.4. Exclusiones del proyecto

No se contempla la posibilidad de que se pueda procesar de forma arbitraria la información histórica. Tampoco se contempla la existencia de una utilidad para definir el procesado de los datos históricos. Además, solo se incluirá la definición del procesado sobre los datos proporcionados.

2.1.5. Restricciones del proyecto

Las restricciones del proyecto son:

- La duración máxima del proyecto es 4 meses.

- Para el procesado de datos históricos se debe utilizar Apache Hadoop.
- Para el procesado de datos en tiempo real se debe utilizar Apache Storm.
- Para la recepción de los datos en tiempo real se debe utilizar Apache Kafka.
- Como sistema de ficheros distribuido se debe utilizar HDFS (Hadoop Distributed File System).

2.1.6. Supuestos del proyecto

Los supuestos del proyecto son:

- Se tendrá acceso a los datos históricos existentes.
- El procesamiento de los datos requeridos estará claramente definido.
- La existencia de un *Data Adquisition Engine* (DAE) que obtendrá los datos de los sensores a través de diferentes protocolos y que los enviará a la cola de mensajes.
- El formato en que el DAE proporciona los datos de los sensores no variará a lo largo del proyecto.

2.2. Gestión de riesgos

Este apartado tiene como objetivo la identificación, categorización y planificación de los riesgos que puedan afectar al proyecto.

Cada riesgo será categorizado por su probabilidad de ocurrencia y por su impacto.

La **probabilidad de ocurrencia** es una estimación de la probabilidad de que un riesgo se materialice. La escala utilizada para este caso está definida por los siguientes niveles:

- **Alta**. Probabilidad estimada mayor o igual a 70 %.
- **Media**. Probabilidad estimada entre el 35 % y el 70 %.

- **Baja**. Probabilidad estimada menor o igual al 35 %.

El **impacto** es el nivel de repercusión para el proyecto producido por la materialización de un riesgo. La escala utilizada en este caso es la siguiente:

- **Catastrófico**. El impacto podría causar la cancelación del proyecto.

- **Serio**. El impacto podría causar una disminución de la calidad final del producto y/o causar variaciones importantes en la planificación.

- **Tolerable**. El impacto podría suponer pequeñas variaciones en la planificación. La calidad final del producto no se vería afectada.

Para cada uno de los riesgos se indica una estrategia a seguir. Para aquellos riesgos a los que se la va a hacer seguimiento, se especifica también un plan de contingencia y/o un plan de prevención.

En el plan de contingencia se explica qué acciones se realizarán una vez que se haya materializado el riesgo. En el plan de prevención se detallan las acciones que se llevarán a cabo para intentar evitar que aparezca un riesgo.

2.2.1. Especificación de riesgos

A continuación se muestran las especificaciones formales de los riesgos. Dicha especificación está formada por: un identificador, un nombre, una descripción, la probabilidad de ocurrencia, el impacto, la estrategia a seguir y el plan de prevención y/o el plan de contingencia.

ID	R-1
Nombre	Retraso en la planificación
Descripción	Una planificación poco realista provoca retrasos en el desarrollo del proyecto.
Probabilidad de ocurrencia	Alta
Impacto	Catastrófico
Estrategia	El plan de prevención se ejecutará al principio del proyecto. Si el riesgo se materializa se ejecutará el plan de contingencia.
Plan de prevención	Establecer reuniones semanales en las que se evalúe el progreso actual del proyecto.
Plan de contingencia	Reducir el alcance del proyecto y eliminar los requisitos que no sean vitales.

ID	R-2
Nombre	Problemas a la hora de comprender las tecnologías
Descripción	Como el alumno no dispone de un gran conocimiento de todas las tecnologías que se van a utilizar, es posible que se presenten retrasos debido a la falta de familiaridad con dichas tecnologías.
Probabilidad de ocurrencia	Media
Impacto	Tolerable
Estrategia	El plan de prevención se ejecutará al principio del proyecto. Si el riesgo se materializa se ejecutará el plan de contingencia.
Plan de prevención	Se incluirá en la planificación un plan de formación para cada una de las tecnologías.
Plan de contingencia	Solicitar reuniones con los tutores para la resolución de las dudas.

ID	R-3
Nombre	Cambios en los requisitos
Descripción	Es posible que en algún punto del desarrollo del proyecto, los requisitos varíen.
Probabilidad de ocurrencia	Baja
Impacto	Serio
Estrategia	Se acepta el riesgo debido a que las probabilidades de que se materialice son bajas.

ID	R-4
Nombre	Baja de los tutores
Descripción	Es posible que por motivos de diversa índole, alguno de los tutores se vea obligado a abandonar el proyecto
Probabilidad de ocurrencia	Baja
Impacto	Tolerable
Estrategia	Se acepta el riesgo debido a que las probabilidades de que se materialice son bajas.

ID	R-5
Nombre	No es posible procesar los datos en tiempo real
Descripción	No se dispone de los recursos necesarios para poder procesar la cantidad de datos que llegan por segundo.
Probabilidad de ocurrencia	Media
Impacto	Serio
Estrategia	Si el riesgo se materializa se ejecutará el plan de contingencia.
Plan de contingencia	Incrementar el nivel de paralelismo añadiendo más recursos.

ID	R-6
Nombre	Pérdida de información asociada al proyecto
Descripción	Cabe la posibilidad que durante el desarrollo del proyecto se pierda información debido a un fallo en el disco donde esté almacenado.
Probabilidad de ocurrencia	Media
Impacto	Serio
Estrategia	El plan de prevención se ejecutará semanalmente. Si el riesgo se materializa se ejecutará el plan de contingencia.
Plan de prevención	La realización de backups de toda la información asociada al proyecto.
Plan de contingencia	Se recuperará la información desde el último backup disponible y se tendrá que rehacer todo el trabajo desde ese punto.

ID	R-7
Nombre	Inutilización del equipo de trabajo
Descripción	El equipo utilizado durante el desarrollo podría dejar de funcionar por causas de diversa naturaleza, como por ejemplo, fallos eléctricos debido a los Sistemas de Alimentación Ininterrumpida del edificio.
Probabilidad de ocurrencia	Media
Impacto	Serio
Estrategia	Si el riesgo se materializa se ejecutará el plan de contingencia.
Plan de contingencia	El alumno utilizará su propio portátil para continuar con el desarrollo. En él deberá desplegar de nuevo toda la arquitectura e instalar el software de desarrollo necesario. Además, recuperará de los backups realizados toda la documentación y código desarrollado.

ID	R-8
Nombre	No se consigue acceso a un cluster
Descripción	No se dispone de acceso a los recursos necesarios para poner el sistema en producción.
Probabilidad de ocurrencia	Alta
Impacto	Tolerable
Estrategia	Si en el momento de comenzar la parte de preparación de la infraestructura y despliegue de las tecnologías no se tiene acceso a un cluster, se ejecutará el plan de contingencia.
Plan de contingencia	Se virtualizarán los recursos para la construcción del prototipo.

2.3. Gestión de la configuración

En este apartado se explica el proceso de gestión de la configuración seguido durante el desarrollo del proyecto con el fin de asegurar la integridad de los productos desarrollados. Esto incluye la identificación de los elementos de configuración, las herramientas utilizadas y la nomenclatura empleada.

Una buena gestión de la configuración permite ser más productivo y eficiente. Alguna de sus ventajas son: evitar la pérdida de versiones anteriores de un determinado documento, tener disponible en todo momento la última versión de cualquier documento, prevenir modificaciones sobre una versión que no es la más reciente, etc.

Cabe destacar que en este proyecto solo hay un único desarrollador, que es el propio alumno, y es el que va a realizar todos los cambios. Esto permite utilizar una gestión de la configuración muy ligera. Por tanto, el principal objetivo del proceso de gestión utilizado en este proyecto es el mantenimiento correcto de las diferentes versiones.

2.3.1. Elementos de configuración

Los ECS identificados son:

- Memoria del proyecto

- Código fuente - Importación datos HDFS
- Código fuente - Procesado datos
- Código fuente - Topología Storm
- Código fuente - Funciones Cassandra
- Código fuente - Página web
- Planificación del proyecto
- Diagramas con los diseños

Para la realización de la memoria del proyecto se ha utilizado el editor de LaTeX online llamado *Overleaf* [2]. Uno de los motivos por los que ha optado por esta herramienta es que utiliza Git para el control de versiones del propio documento. De este modo, es muy fácil etiquetar versiones, deshacer cambios, etc. Otro de los motivos es que se pueden poner comentarios de una manera rápida sobre el propio documento, lo que permite que los tutores puedan revisar la memoria y comentar los problemas que encuentren de manera cómoda.

Para todo el código fuente desarrollado, los diagramas que muestran el diseño de cada una de las partes del sistema y la planificación del proyecto se ha utilizado la herramienta de control de versiones Git. El repositorio empleado se creó en la plataforma GitLab [3] del *CiTIUS*.

En el caso de la planificación, a pesar de que forma parte del repositorio, se utiliza la propia herramienta para gestionar las líneas base. Esto quiere decir, que en el propio fichero de la planificación, están todas las líneas base creadas, lo que permite ver los cambios entre ellas rápidamente.

2.3.2. Líneas base

Se creará una línea base al finalizar cada incremento. Para ello, se hará uso de los *tags* de Git. Cada *tag* representará una línea base y serán denominadas de la siguiente forma:

```
LB-<númeroLíneaBase>_<aaMMdd>
```

Un ejemplo es el siguiente:

```
LB-1_160510
```

Ésta sería la línea base número 1 y fue creada el 10 de mayo de 2016.

Entre línea base y línea base habrá ficheros que no están actualizados. Por ejemplo, si se acaba de realizar el diseño de una de las capas y se va a proceder con la implementación, es evidente que en ese momento el diseño no representa la implementación actual. Esto es totalmente normal entre línea base y línea base. Este problema se soluciona en el momento en el que se crea una. En ese instante, se verifica que cada uno de los ficheros haya sido actualizado y esté correcto. Una vez comprobado, se crea la línea base correspondiente.

En la tabla siguiente se muestran los entregables de cada línea base:

Línea base	**ECS**
1	Memoria del proyecto Planificación del proyecto Diagramas con los diseños Código fuente - Importación datos HDFS Código fuente - Topología Storm
2	Memoria del proyecto Planificación del proyecto Diagramas con los diseños Código fuente - Importación datos HDFS Código fuente - Topología Storm Código fuente - Procesado datos
3	Memoria del proyecto Planificación del proyecto Diagramas con los diseños Código fuente - Importación datos HDFS Código fuente - Topología Storm Código fuente - Procesado datos Código fuente - Funciones Cassandra Código fuente - Página web

2.4. Metodología de desarrollo

Para poder abordar el proyecto de forma exitosa es muy importante definir cual va a ser la metodología utilizada.

Este proyecto tiene la característica de ser desarrollado de forma individual. Esto implica que no existen equipos de trabajo. Por tanto, no es necesario realizar una repartición de tareas, ni establecer reuniones para planificar el trabajo dentro del equipo, ni ningún tipo de tarea relacionada con la coordinación entre varias personas. Otra característica de este proyecto es que la arquitectura que se va a desplegar está formada por capas. Esto permite ver el proyecto como un conjunto de subsistemas, cada uno con una funcionalidad distinta.

Por otra parte, tanto los objetivos del proyecto como los requisitos tienen muy pocas probabilidades de que varíen. Esto es debido a que no existe un *cliente* que pueda cambiar de opinión durante el desarrollo.

Una restricción muy importante del proyecto es el plazo de entrega del proyecto. Para conseguir tener terminado el proyecto a tiempo es necesario que la metodología permita ver los resultados del trabajo cuanto antes. Esto está muy relacionado con el riesgo *R-3 (Incompatibilidades entre las diferentes tecnologías)*. Si hay problemas a la hora de acoplar las tecnologías es muy importante detectarlo cuanto antes para intentar tener un margen de maniobra para poder solucionar los problemas que pudieran surgir.

Dadas las restricciones y características anteriores, se ha decidido utilizar como ciclo de vida el modelo incremental. La idea es realizar 3 incrementos y, en cada uno de ellos desarrollar y desplegar una parte de la arquitectura final. En el primer incremento se abordarán dos capas: la capa de adquisición y la capa de procesado de datos en tiempo real. El motivo de abordar estas dos capas en el primer incremento es que están muy relacionadas y no se puede realizar completamente una sin la otra. En el segundo incremento se abordará la capa de procesamiento de datos históricos. Y, en el tercer incremento, se abordará la capa de diseminación.

Como se ha comentado en el apartado de gestión de la configuración, la finalización de cada incremento genera ECS y una nueva línea base.

Además, el realizar el proyecto de forma incremental permite ir realizando

pruebas en cada iteración. Es decir, en el primer incremento ya se prueba la interoperabilidad entre dos de las capas. Si todo funciona correctamente, se despliega la tercera capa y se realizan pruebas para comprobar que las tres capas no tienen ningún problema. Y lo mismo con la última capa.

2.5. Planificación temporal

En este apartado se muestra, por un lado, el plan de ejecución realizado al principio del proyecto y, por otro lado, el plan de ejecución que se siguió finalmente.

2.5.1. Plan inicial

La planificación realizada al comienzo del proyecto puede verse en la Figura 2.1. Ésta está formada por 6 bloques que serán detallados en los subapartados siguientes. En ningún punto se incluyó una tarea denominada documentación ya que se considera que todas las tareas llevan implícita una parte de documentación.

Una opción para mostrar esta tarea de documentación sería crear una tarea que durara todo el proyecto. A ésta se le dedicaría el 10 % del tiempo y el 90 % al resto de las tareas. Sin embargo, esta opción se descartó porque se considera que no aporta nada a la planificación.

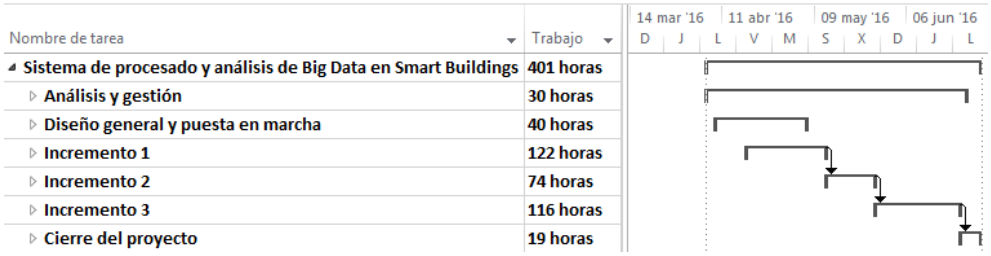


Figura 2.1: Planificación inicial

Análisis y gestión

En este bloque se realizan todas las tareas de gestión y análisis. Como se puede ver en la Figura 2.2, las tareas de análisis y gestión están solapadas. El motivo

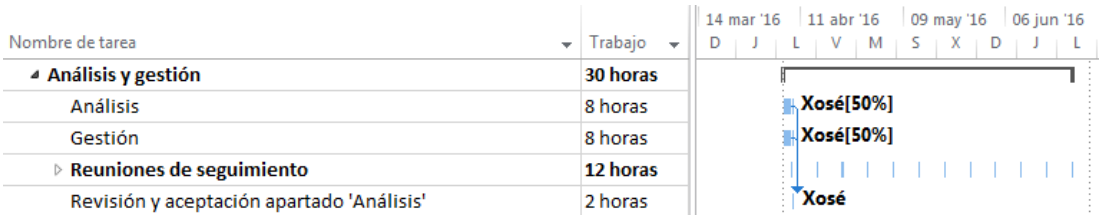


Figura 2.2: Planificación inicial - Análisis y gestión

es que están muy relacionadas entre ellas y se ha decido hacerlas juntas. Además de esas dos tareas, también están las reuniones de seguimiento. Éstas están planificadas para ser realizadas todos los viernes con el objetivo de comprobar el progreso del proyecto. Por último está la tarea de revisión y aceptación de la parte de análisis. Es importante destacar que la parte de gestión está presente durante todo el proyecto, tal y como se puede ver en la Figura 2.1.

Diseño general y puesta en marcha

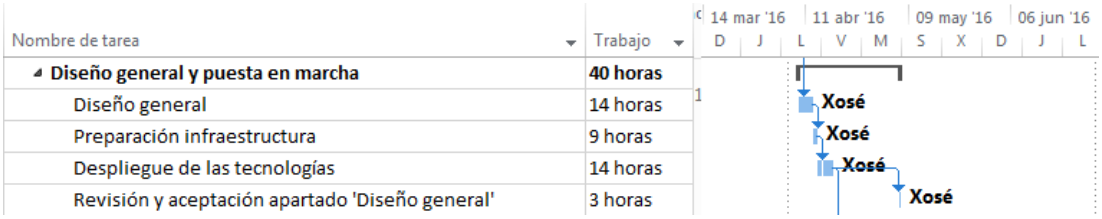


Figura 2.3: Planificación inicial - Diseño general y puesta en marcha

La primera tarea de este bloque es realizar un diagrama de alto nivel de toda la arquitectura. Esto incluye decidir cómo se va a desplegar cada una de las tecnologías, en cuantas máquinas, qué servicios en cada máquina, etc. Una vez terminado el diseño general, se prepara la infraestructura y se procede a la instalación de cada una de las tecnologías. Este bloque es cerrado con la tarea de revisión y aceptación el diseño realizado.

Incremento 1

En este incremento se abordan dos capas de la arquitectura, la capa de adquisición y la capa de procesamiento de datos en tiempo real. La primera tarea dentro de la capa de adquisición es estudiar el funcionamiento de cada una de

Nombre de tarea	Trabajo
Incremento 1	**122 horas**
Capa de adquisición	**43 horas**
Estudio Apache Kafka, paradigma MapReduce y JSON	17 horas
Migración de los datos al Hadoop Distributed File System	**26 horas**
Diseño programa MapReduce para la importación de los datos	16 horas
Codificación programa MapReduce para la importanción de los datos	10 horas
Capa de procesamiento de datos en tiempo real	**55 horas**
Estudio de Apache Storm	20 horas
Diseño de la topología	21 horas
Codificación de la topología	14 horas
Pruebas (Diseño/Codificación/Ejecución/Correción fallos encontrados)	21 horas
Revisión y aceptación 'Incremento 1'	3 horas

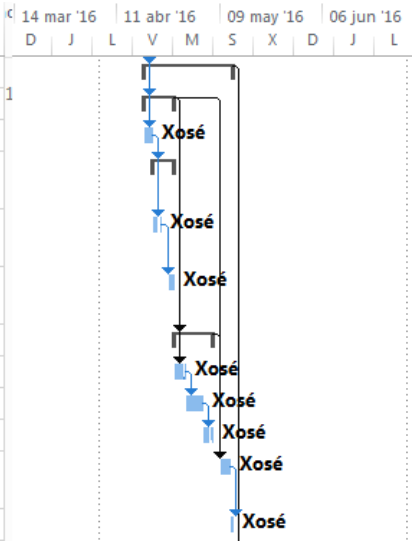


Figura 2.4: Planificación inicial - Incremento 1

las tecnologías que se van a utilizar en esta capa. La segunda tarea es realizar la migración de los datos históricos actuales a la nueva arquitectura. Eso con respecto a la primera capa. Con respecto a la segunda, ésta comienza con una parte de estudio de la tecnología a utilizar y continúa con el diseño y codificación de la topología para Apache Storm. Este incremento finaliza con la realización de las pruebas y con la revisión y aceptación de las dos capas desplegadas.

Incremento 2

Nombre de tarea	Trabajo
Incremento 2	**74 horas**
Capa de procesamiento de datos históricos	**49 horas**
Estudio de Apache Cassandra	14 horas
Diseño de los programas MapReduce	21 horas
Codificación de los programas MapReduce	14 horas
Pruebas (Diseño/Codificación/Ejecución/Correción fallos encontrados)	21 horas
Revisión y aceptación 'Incremento 2'	4 horas

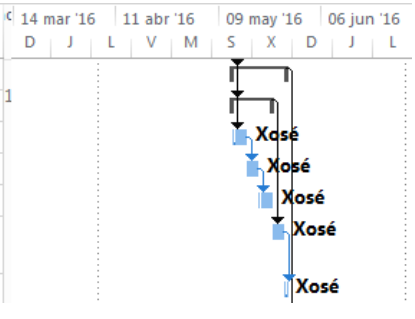


Figura 2.5: Planificación inicial - Incremento 2

En este incremento se aborda la capa de procesamiento de datos históricos. Al igual que en las capas anteriores, ésta comienza con una parte de estudio de las tecnologías que se van a utilizar seguida del diseño y la codificación de los

programas MapReduce necesarios. Este incremento finaliza con la realización de las pruebas y con la revisión y aceptación de la capa desplegada.

Incremento 3

Nombre de tarea	Trabajo
Incremento 3	**116 horas**
Capa de diseminación	**91 horas**
Selección y estudio del framework web a utilizar	21 horas
Back-End	**49 horas**
Desarrollo del Back-End	21 horas
Unificación de las vistas de tiempo real e histórico	28 horas
Front-End	14 horas
Despliegue de la web	7 horas
Pruebas (Diseño/Codificación/Ejecución/Correción fallos encontrados)	21 horas
Revisión y aceptación 'Incremento 3'	4 horas

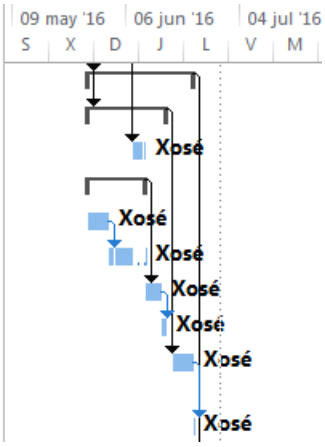


Figura 2.6: Planificación inicial - Incremento 3

En este incremento se aborda la última la capa, la capa de diseminación. La primera tarea consiste en la selección y el estudio de un framework web. La segunda tarea es el desarrollo del Back-End y de las funciones o programas necesarios para unificar las vistas en tiempo real e históricas. A continuación se desarrolla toda la parte del Front-End. Este incremento, al igual que los anteriores, finaliza con la realización de las pruebas y con la revisión y aceptación de la capa desplegada.

Cierre del proyecto

Nombre de tarea	Trabajo
Cierre del proyecto	**19 horas**
Revisión de la documentación y del código	14 horas
Impresión de la memoria	0 horas
Preparación de todo el material a entregar	5 horas
Entrega final	0 horas

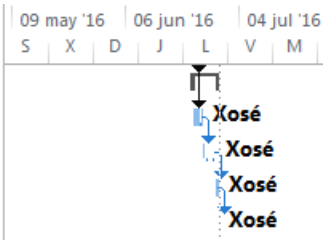


Figura 2.7: Planificación inicial - Cierre del proyecto

Este último bloque incluye la revisión de la documentación y de todo el código desarrollado, la impresión de la memoria, la preparación y del material a entregar y, por último, la propia entrega del proyecto.

2.5.2. Plan final

En la Figura 2.8 se puede ver cuál fue realmente el plan de ejecución seguido. Como se puede ver en la mitad derecha de la figura, se produjeron retrasos. Éstos fueron debido a que determinadas tareas llevaron más tiempo del previsto y a la aparición de nuevas tareas que no se habían sido contempladas.

Figura 2.8: Planificación final

Análisis y gestión

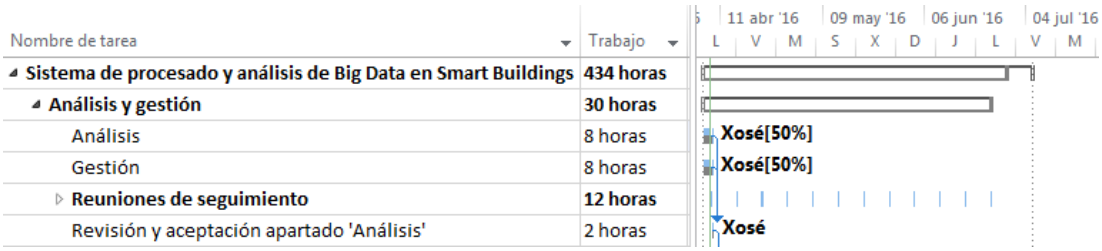
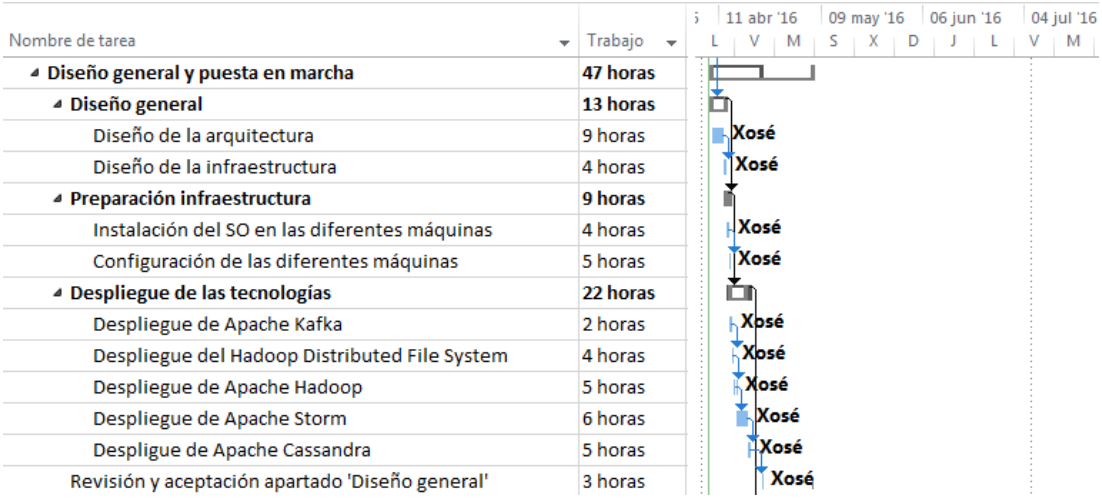


Figura 2.9: Planificación final - Análisis y gestión

En este bloque no se produjeron retrasos significantes.

Diseño general y puesta en marcha

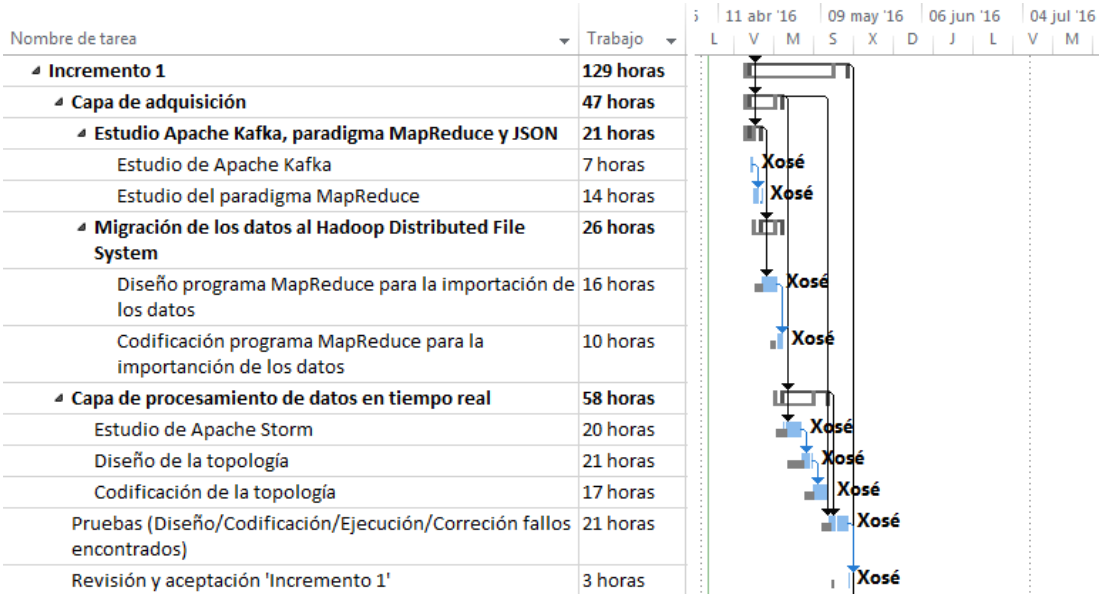
En este bloque las tareas duraron más de lo previsto debido a la aparición de algunos problemas a la hora de instalar alguna de las tecnologías.



Nombre de tarea	Trabajo
Diseño general y puesta en marcha	**47 horas**
Diseño general	**13 horas**
Diseño de la arquitectura	9 horas
Diseño de la infraestructura	4 horas
Preparación infraestructura	**9 horas**
Instalación del SO en las diferentes máquinas	4 horas
Configuración de las diferentes máquinas	5 horas
Despliegue de las tecnologías	**22 horas**
Despliegue de Apache Kafka	2 horas
Despliegue del Hadoop Distributed File System	4 horas
Despliegue de Apache Hadoop	5 horas
Despliegue de Apache Storm	6 horas
Despligue de Apache Cassandra	5 horas
Revisión y aceptación apartado 'Diseño general'	3 horas

Figura 2.10: Planificación final - Diseño general y puesta en marcha

Incremento 1



Nombre de tarea	Trabajo
Incremento 1	**129 horas**
Capa de adquisición	**47 horas**
Estudio Apache Kafka, paradigma MapReduce y JSON	**21 horas**
Estudio de Apache Kafka	7 horas
Estudio del paradigma MapReduce	14 horas
Migración de los datos al Hadoop Distributed File System	**26 horas**
Diseño programa MapReduce para la importación de los datos	16 horas
Codificación programa MapReduce para la importancián de los datos	10 horas
Capa de procesamiento de datos en tiempo real	**58 horas**
Estudio de Apache Storm	20 horas
Diseño de la topología	21 horas
Codificación de la topología	17 horas
Pruebas (Diseño/Codificación/Ejecución/Correción fallos encontrados)	21 horas
Revisión y aceptación 'Incremento 1'	3 horas

Figura 2.11: Planificación final - Incremento 1

En este incremento las tareas de estudio y alguna de codificación y diseño llevaron más tiempo del previsto. Esto supuso 7 horas más de trabajo en este incremento.

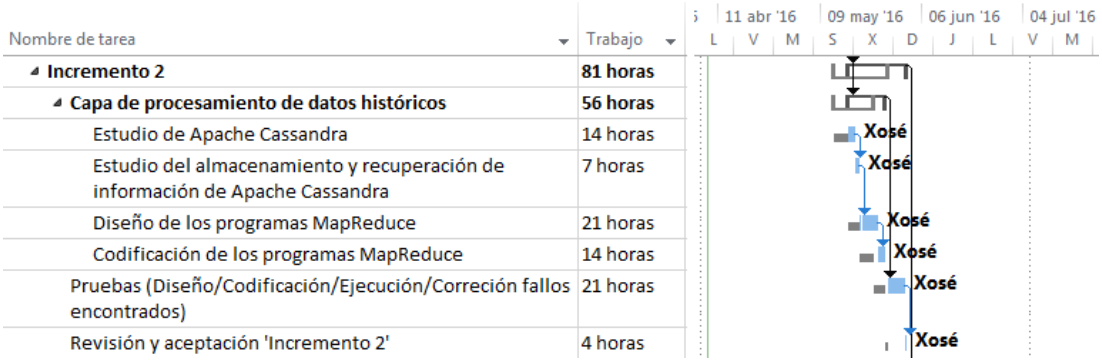


Figura 2.12: Planificación final - Incremento 2

Incremento 2

En este incremento apareció una nueva tarea que no estaba contemplada en el plan inicial que es: *Estudio del almacenamiento y recuperación de información de Apache Cassandra*. Esto sumado a algún otro retraso supuso 6 horas de trabajo adicionales.

Incremento 3

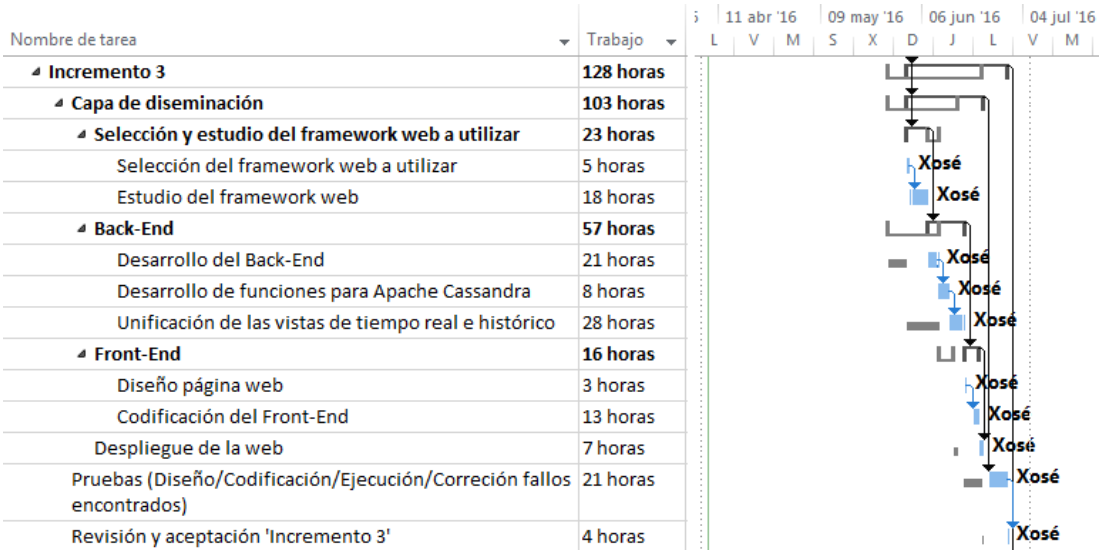


Figura 2.13: Planificación final - Incremento 3

Este incremento supuso más de 12 horas de trabajo adicional. La parte que tuvo más dificultades fue el Back-End de la web. También influyó el tiempo de

estudio del framework, que fue mayor que el esperado.

Cierre del proyecto

Figura 2.14: Planificación final - Cierre del proyecto

En la parte de cierre del proyecto no hay ninguna diferencia apreciable con respecto a lo planificado inicialmente.

2.6. Análisis de costes

En este apartado se analiza el coste económico proveniente de la realización de este proyecto. En este análisis se pueden distinguir dos tipos de costes:

- **Directos.** Están exclusivamente vinculados al proyecto. Un ejemplo de este tipo de costes son los salarios de los miembros del equipo del proyecto.
- **Indirectos.** Se reparten entre diferentes proyectos. Generalmente se calculan como un porcentaje de los costes directos. Un ejemplo de este tipo de costes es el consumo de agua.

2.6.1. Costes directos

El primer coste directo es el salario correspondiente al desarrollador. No se incluye a los tutores ya que en este proyecto ellos asumen el rol de cliente. Para el salario del desarrollador se tomará el valor proporcionado por la empresa *Vitae Consultores* en su *Guía Salarial del Sector TIC en Galicia 2015-2016* [4]. En ella, un analista programador en Java con una experiencia de 0-2 años, es decir, Junior, tiene un salario medio de 16.000 € brutos anuales. Se asumirá un coste de la seguridad social de un 31 % sobre el salario bruto. El coste por hora a la

empresa suponiendo 14 pagas al año, 8 horas diarias y 20 días laborables al mes sería:

Bruto anual	Seg. Social	Total anual	Total mensual	Coste/hora
16.000 €	4.960 €	20.960 €	1.497 €	9,36 €

La duración de este proyecto es de 412.5 horas, por lo que supondría un coste de 3.861 €.

Para el desarrollo de este proyecto se ha utilizado un equipo de sobremesa adquirido por el *CiTiUS* por un precio de 1.200 €. La vida media del ordenador se estima en unos 6 años. Teniendo en cuenta que este proyecto tiene una duración de 4 meses, el coste del equipo en el proyecto es:

Meses de vida	Meses de uso	Coste total	Coste en el proyecto
72	4	1.200 €	66,67 €

Todo el software utilizado para este proyecto tiene la característica de ser software libre. La única excepción es el Microsoft Project. En este caso se ha utilizado la licencia de prueba, por lo que no supone ningún coste a mayores.

Por tanto, a modo de resumen:

Elemento	Coste
Analista programador en Java	3.891 €
Equipo de sobremesa	66,67 €
Total costes directos	**3.958 €**

2.6.2. Costes indirectos

Para el cálculo de los costes indirectos se van a seguir las políticas de la USC [5] y se va a aplicar un 20 % sobre los costes directos. Esto implica que los costes indirectos son **792 €**.

2.6.3. Costes totales

Sumando los costes directos e indirectos, se obtiene un coste total del proyecto de **4.750 €**.

Capítulo 3

Análisis de requisitos

En este capítulo se aborda la parte de análisis del proyecto. Ésta está formada por la especificación de los requisitos y por la especificación de los casos de uso.

La forma de especificar un sistema tiene una gran influencia en la calidad de la solución implementada finalmente. Es decir, el trabajar con especificaciones erróneas o incompletas llevan a una disminución de la calidad y completitud del software. Por tanto, la fase de análisis es muy importante y debe ser abordada con minuciosidad.

3.1. Especificación de requisitos

Los requisitos serán agrupados en: requisitos funcionales y requisitos no funcionales. Cada de uno de ellos tendrá asignado un identificador, que será **RF-x** para los requisitos funcionales y **RNF-x** para los requisitos no funcionales, siendo x el número que le corresponda. También se aportará un nombre y una descripción. Además, se les asignará una importancia que podrá ser **Vital**, si se considera imprescindible para el proyecto, o **Deseable**, si no es indispensable pero mejoraría la calidad del producto final.

3.1.1. Requisitos funcionales

ID	RF-1
Nombre	Obtención de los datos
Descripción	El sistema deberá obtener los datos generados por la red de sensores en tiempo real.
Importancia	Vital

ID	RF-2
Nombre	Almacenamiento de los datos
Descripción	El sistema deberá guardar los datos obtenidos en su forma original.
Importancia	Vital

ID	RF-3
Nombre	Procesado de los datos
Descripción	El sistema deberá procesar todos los datos recibidos de los sensores y generar las vistas oportunas.
Importancia	Vital

ID	RF-4
Nombre	Almacenamiento de las vistas generadas
Descripción	El sistema deberá almacenar las vistas generadas en una base de datos.
Importancia	Vital

ID	RF-5
Nombre	Consulta de datos
Descripción	El sistema deberá permitir consultar y visualizar los datos del sistema a partir de las vistas almacenadas.
Importancia	Vital

3.1.2. Requisitos no funcionales

ID	RNF-1
Nombre	Tolerancia a fallos hardware
Descripción	El sistema deberá ser tolerante a fallos causados en sus elementos hardware.
Importancia	Vital

ID	RNF-2
Nombre	Cantidad de datos de entrada
Descripción	La red de sensores del *CiTIUS* genera 667 señales procedentes de los sensores cada 10 segundos. El sistema deberá ser capaz hacer frente a esa cantidad de información.
Importancia	Vital

ID	RNF-3
Nombre	Respuesta en tiempo real
Descripción	El sistema deberá actualizar el estado y dar respuesta en menos de 10 segundos a las consultas predefinidas sobre todos los datos generados por el sistema.
Importancia	Vital

ID	RNF-4
Nombre	Escalabilidad del sistema
Descripción	El sistema deberá ser escalable tanto en la capa de procesado como de almacenamiento. Se entiende por escalabilidad la habilidad del sistema para mantener el rendimiento cuando el volumen de datos almacenados o el número de señales de entrada aumenta.
Importancia	Vital

ID	RNF-5
Nombre	Generalización del sistema
Descripción	La arquitectura del sistema podrá ser empleada en diferentes aplicaciones. En este caso, se utiliza para *Smart buildings*, pero el modo en el que se almacenen y traten los datos tiene que poder ser aplicable a otros tipos de aplicaciones Big Data.
Importancia	Deseable

ID	RNF-6
Nombre	Consultas Ad-hoc
Descripción	Partiendo de la premisa de que los grandes volúmenes de datos suelen contener información y relaciones no evidentes, el sistema deberá estar pensado para soportar la implementación de consultas arbitrarias sobre los elementos almacenados, de forma que en el futuro pueda ser extendido con facilidad.
Importancia	Vital

ID	RNF-7
Nombre	Tolerancia a errores humanos
Descripción	El sistema podrá recuperarse de implementaciones erróneas de los algoritmos de procesado de los datos. El almacenamiento de los datos originales de forma permanente e inmutable nos permitirá procesarlos nuevamente y revertir cualquier cambio realizado debido a un error previo.
Importancia	Vital

3.2. Casos de uso

Los casos de uso permiten especificar a alto nivel cuál es el comportamiento de un sistema. Un caso de uso representa una operación o una tarea específica que se realiza tras la orden de alguno de los actores, o bien tras la invocación desde otro caso de uso.

3.2.1. Actores

La descripción de los actores identificados, siguiendo una especificación formal, se presenta a continuación:

ID	A-1
Nombre	Usuario
Descripción	Este actor representa al usuario que interacciona con la aplicación.

ID	A-2
Nombre	Timer
Descripción	Este actor representa a un temporizador que se encarga de lanzar tareas periódicas.

3.2.2. Especificación de casos de uso

En la Figura 3.1 se puede ver el diagrama de casos de uso realizado. Además, cada caso de uso será detallado siguiendo una especificación formal formada por los siguientes campos: ID, Nombre, Requisitos asociados, Descripción, Precondición, Secuencia normal, Secuencias alternativas (en caso de haberlas), Postcondicición e Importancia.

La importancia de los casos de uso podrá ser:

- **Alta**. El caso de uso es imprescindible para el producto final.
- **Media**. El caso de uso amplía considerablemente las funcionalidades del producto final, pero no es indispensable.
- **Baja**. El caso de uso complementa a otros casos de uso, y su falta no causa una gran pérdida de funcionalidades.

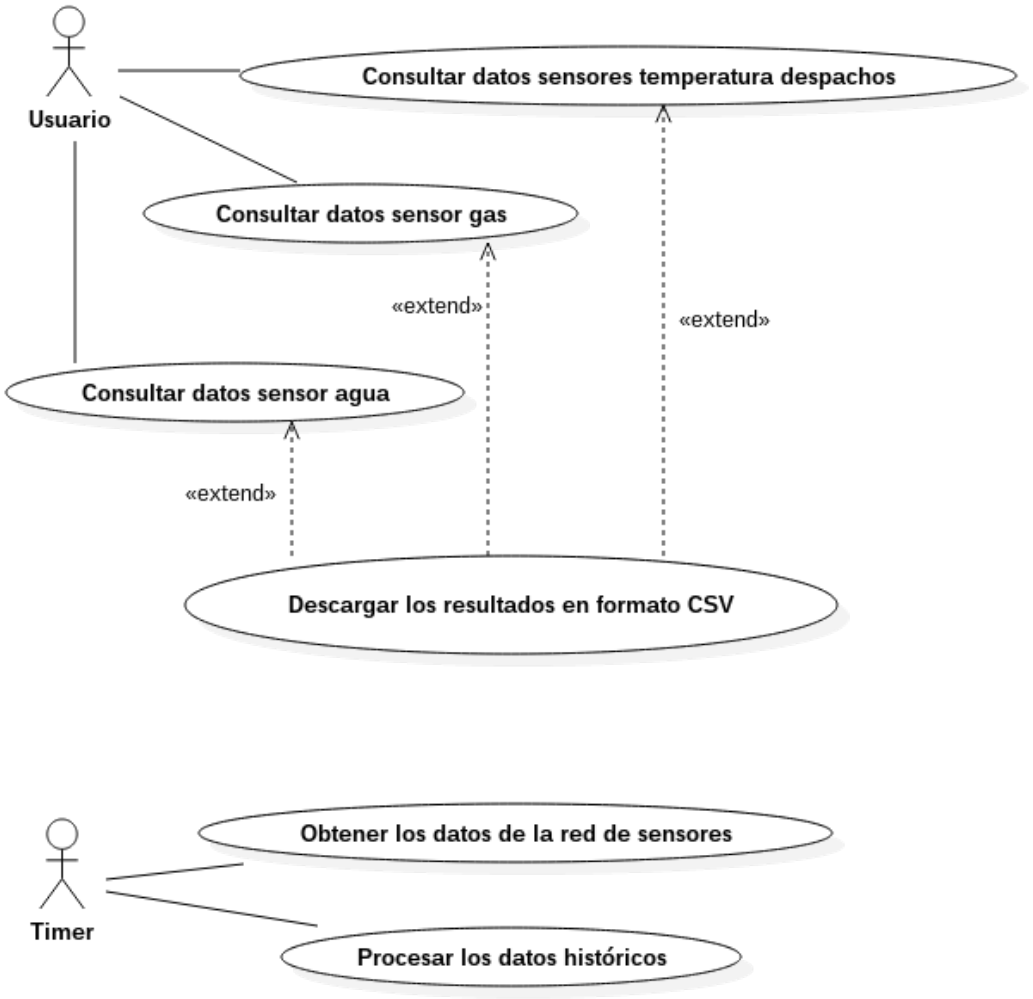


Figura 3.1: Diagrama de casos de uso

ID	CAU-1	
Nombre	Consultar datos sensor agua	
Requisitos asociados	[RF-5] Consulta de datos	
Descripción	El sistema deberá permitir al usuario realizar consultas acerca del sensor del consumo de agua.	
Precondición	El usuario ha seleccionado *Agua* como tipo de sensor.	
Secuencia normal	Paso	Acción
	1	El usuario especifica la fecha de inicio.
	2	El usuario especifica la fecha de fin.
	3	El usuario selecciona el intervalo de tiempo.
	4	El usuario pulsa el botón *Enviar*.
Secuencia alternativa	Paso	Acción
	4	El sistema detecta que la fecha de fin es anterior a la fecha de inicio.
	5	El sistema muestra un mensaje de error.
Postcondición	Si los datos introducidos son válidos, se le mostrará al usuario una nueva página en donde se podrá ver en una gráfica el resultado de su consulta. En cambio, si los datos introducidos no son válidos, se le mostrará un mensaje de error indicando la causa y no se lanzará la consulta.	
Importancia	Alta	

ID	CAU-2	
Nombre	Consultar datos sensor gas	
Requisitos asociados	[RF-5] Consulta de datos	
Descripción	El sistema deberá permitir al usuario realizar consultas acerca del sensor del consumo de gas.	
Precondición	El usuario ha seleccionado *Gas* como tipo de sensor.	
Secuencia normal	Paso	Acción
	1	El usuario especifica la fecha de inicio.
	2	El usuario especifica la fecha de fin.
	3	El usuario selecciona el intervalo de tiempo.
	4	El usuario pulsa el botón *Enviar*.
Secuencia alternativa	Paso	Acción
	4	El sistema detecta que la fecha de fin es anterior a la fecha de inicio.
	5	El sistema muestra un mensaje de error..
Postcondición	Si los datos introducidos son válidos, se le mostrará al usuario una nueva página en donde se podrá ver en una gráfica el resultado de su consulta. En cambio, si los datos introducidos no son válidos, se le mostrará un mensaje de error indicando la causa y no se lanzará la consulta.	
Importancia	Alta	

ID	CAU-3	
Nombre	Consultar datos sensores temperatura despachos	
Requisitos asociados	[RF-5] Consulta de datos	
Descripción	El sistema deberá permitir al usuario realizar consultas acerca de los sensores de temperatura de los despachos.	
Precondición	El usuario ha seleccionado *Temperatura* como tipo de sensor.	
Secuencia normal	Paso	Acción
	1	El usuario especifica la fecha de inicio.
	2	El usuario especifica la fecha de fin.
	3	El usuario selecciona el intervalo de tiempo.
	4	El usuario selecciona el despacho.
	5	El usuario pulsa el botón *Enviar*.
Secuencia alternativa	Paso	Acción
	5	El sistema detecta que la fecha de fin es anterior a la fecha de inicio.
	6	El sistema muestra un mensaje de error..
Postcondición	Si los datos introducidos son válidos, se le mostrará al usuario una nueva página en donde se podrá ver en una gráfica el resultado de su consulta. En cambio, si los datos introducidos no son válidos, se le mostrará un mensaje de error indicando la causa y no se lanzará la consulta.	
Importancia	Alta	

ID	CAU-4	
Nombre	Descargar los resultados en formato CSV	
Requisitos asociados	[RF-5] Consulta de datos	
Descripción	El sistema deberá permitir al usuario descargar los resultados de su consulta en formato CSV.	
Precondición	El usuario ha realizado una consulta y se encuentra en la página en donde se muestra el resultado de su consulta representado en una gráfica.	
Secuencia normal	Paso	Acción
	1	El usuario pulsa el botón *Descargar los resultados en formato CSV*.
	2	Comienza la descarga de los resultados.
Postcondición	El usuario tiene descargado en su dispositivo el fichero CSV que contiene los resultados de la consulta realizada.	
Importancia	Baja	

ID	CAU-5	
Nombre	Obtener los datos de la red de sensores	
Requisitos asociados	[RF-1] Obtención de los datos, [RF-2] Almacenamiento de los datos	
Descripción	Se recupera una nueva señal de un sensor.	
Precondición	La conexión con la cola de mensajes está activa.	
Secuencia normal	Paso	Acción
	1	El *timer* llama la función encargada de recuperar un dato nuevo.
	2	Se descarga un dato nuevo.
	3	Se almacena el dato recibido.
Postcondición	Se ha recuperado y almacenado el dato siguiente al recogido en la descarga anterior.	
Importancia	Alta	

ID	CAU-6	
Nombre	Procesar los datos del sistema	
Requisitos asociados	[RF-3] Procesado de los datos, [RF-4] Almacenamiento de las vistas generadas	
Descripción	Se procesan todos los datos que están disponibles en el sistema, se generan las vistas históricas oportunas y se actualizan las vistas.	
Precondición	-	
Secuencia normal	Paso	Acción
	1	El *timer* ejecuta el programa encargado de procesar los datos y actualizar las vistas.
	2	Se procesan todos los datos disponibles en el sistema.
	3	Se generan las vistas históricas.
	4	Se almacenan las nuevas vistas, sustituyendo a las anteriores.
	5	Se eliminan los datos de las vistas en tiempo real que ya se encuentren en las nuevas vistas históricas.
Postcondición	Se han actualizado las vistas del sistema.	
Importancia	Alta	

3.2.3. Matriz de trazabilidad

Para finalizar el análisis de requisitos, en la Cuadro 3.1 se muestra la matriz de trazabilidad que indica la correspondencia entre requisitos funcionales y casos de uso.

	CAU-1	**CAU-2**	**CAU-3**	**CAU-4**	**CAU-5**	**CAU-6**
RF-1					X	
RF-2					X	
RF-3						X
RF-4						
RF-5	X	X	X	X		

Cuadro 3.1: Matriz de trazabilidad

Capítulo 4

Descripción de las tecnologías

En este capítulo se describe cada una de las tecnologías que va a ser utilizada para desplegar la arquitectura diseñada.

4.1. Map-Reduce y Apache Hadoop

El paradigma Map-Reduce (MR) [6] puede ser considerado el estándar *de facto* para el procesamiento de grandes conjuntos de datos históricos. Básicamente MR divide problemas complejos en tareas más pequeñas que pueden ser resueltas de forma más sencilla y en paralelo. Este paradigma obtiene su principal ventaja en el hecho de que siempre trata de enviar las operaciones al lugar donde están almacenados los datos y así evitar el movimiento de éstos a través de la red.

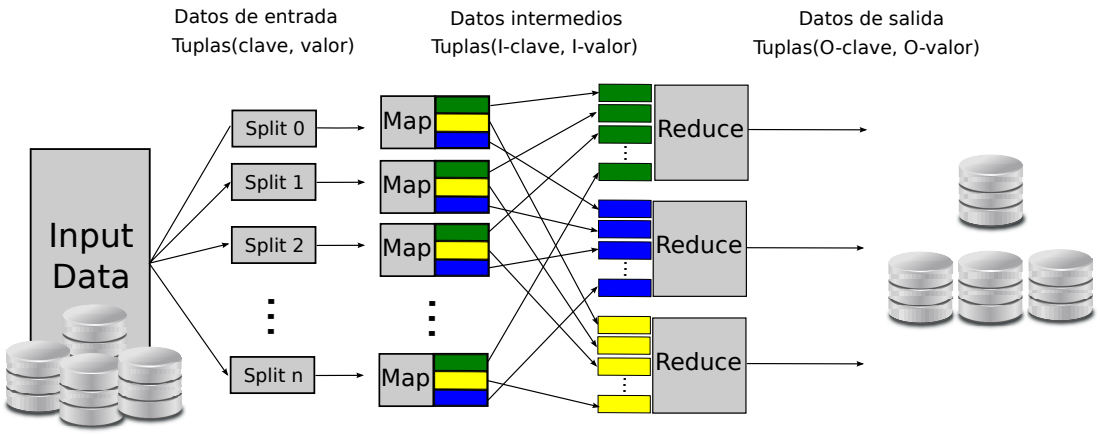


Figura 4.1: Esquema general paradigma MapReduce

La Figura 4.1 muestra un esquema general del paradigma. Un trabajo MR está compuesto principalmente por dos funciones que deben de ser escritas por el desarrollador:

- ***Map***. Esta función toma como entrada un par clave/valor, realiza algún tipo de procesado con los datos de entrada y genera un conjunto intermedio de pares clave/valor, tal y como se puede ver en la Figura 4.2.

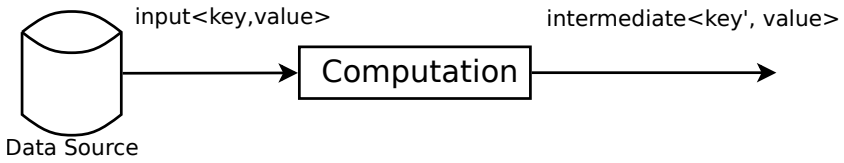


Figura 4.2: Función map

- ***Reduce***. Esta función agrupa los valores del conjunto intermedio que tengan la misma clave y combina sus valores para generar un conjunto más pequeño, tal y como se puede ver en la Figura 4.3.

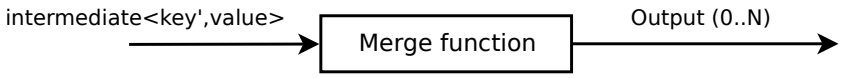


Figura 4.3: Función reduce

Apache Hadoop [7] es un framework open-source que contiene la implementación MR más utilizada. Permite un desarrollo fácil y rápido de aplicaciones distribuidas y tolerantes a fallos para el procesamiento de grandes volúmenes de datos en clusters formados por recursos heterogéneos. Hadoop se encarga de realizar el particionado de los datos de entrada, la comunicación entre las máquinas, manejar los fallos y programar las tareas. De este modo, los desarrolladores sin experiencia en sistemas paralelos y distribuidos pueden desarrollar rápidamente aplicaciones distribuidas.

4.2. Apache Storm

Apache Storm [8, 9] es un sistema distribuido de procesado en tiempo real que tiene como objetivo procesar flujos de datos empleando clusters formados

por recursos heterogéneos. Es un framework de bajo nivel que proporciona una serie de herramientas genéricas para el desarrollo de grafos de computación distribuidos. Estos grafos, llamados topologías, están compuestos por tres elementos principales:

- ***Streams***. Son secuencias ilimitadas de tuplas (clave, valor). Ellos forman los vértices del grafo y simbolizan como fluyen los datos a través de la topología.
- ***Bolts***. Forman los nodos del grafo y contienen toda la lógica de procesamiento que es necesaria aplicar a los streams. Los bolts reciben un número (cualquiera) de streams como entrada, transforman las tuplas que reciben y, finalmente, realizan una de las siguientes acciones:
  - Persisten las tuplas en algún tipo de sistema de almacenamiento de datos.
  - Generan uno o varios streams de datos nuevos que son enviados a otros bolts de la topología.
- ***Spouts***. Son la fuente de los streams. Ellos son los responsables de convertir los datos externos de entrada en secuencias de tuplas que son enviadas a los bolts tal y como dicte la topología.

En la Figura 4.4 se puede ver una pequeña topología de ejemplo.

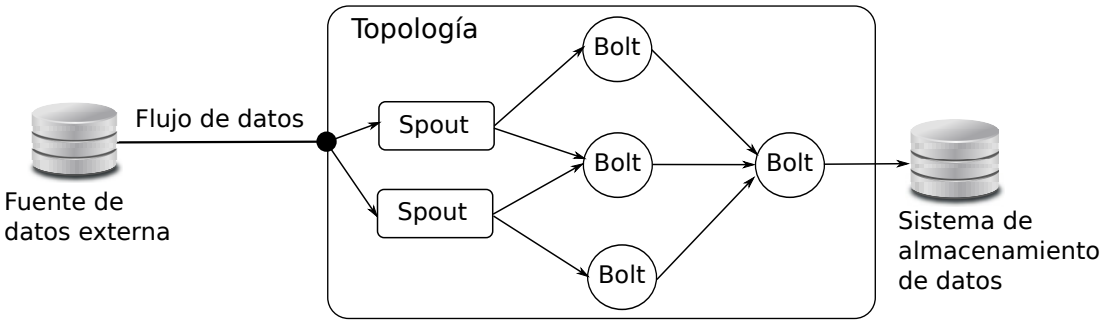


Figura 4.4: Topología de ejemplo

Apache Storm permite configurar paralelismo inicial para cada bolt y spout de la topología. Además, permite alterar el nivel de paralelismo en tiempo de

ejecución. Apache Storm es el responsable de distribuir los hilos de ejecución necesarios sobre el cluster para conseguir ejecutarse correctamente.

El framework también es capaz de garantizar que cada mensaje que salga de un spout va a ser totalmente procesado. Para proporcionar esta garantía, Apache Storm añade automáticamente un bolt a la topología que se encarga monitorizar las tuplas que están siendo procesadas. Una vez que una tupla recorre todo el grafo y está totalmente procesada, todos los nodos que han participado en el procesamiento de dicha tupla son notificados. Sin embargo, si la tupla no está totalmente procesada, va a ser etiquetada como fallida y se le notificará a los nodos implicados. Este mecanismo de vuelta atrás puede, en algún momento, alcanzar el spout que envió la tupla inicialmente. De este modo, el spout sería el responsable de retirar o reenviar la tupla fallida de acuerdo con la lógica de la topología.

4.3. Apache ZooKeeper

Apache ZooKeeper [10] es un servicio centralizado para coordinar procesos distribuidos que típicamente es empleado para almacenar información de configuración o de estado de dichos procesos.

Está formado por un espacio de nombres jerárquico compartido de registros de datos (*znodes*). Todos los datos se mantienen en memoria y, cada *znode* tiene limitada la cantidad de información que puede tener. Además, el servicio está replicado sobre un conjunto de máquinas. Un ejemplo de como sería el espacio de nombres es el que se muestra en la Figura 4.5.

4.4. Apache Kafka

Apache Kafka [11, 12] es un sistema de mensajería publica-suscribe distribuido, que está diseñado para ser rápido, escalable y duradero.

Como otros muchos sistemas de mensajería publica-suscribe, Kafka mantiene diferentes colas (*topics*). Los productores escriben información en una cola y, los consumidores, leen la información de las colas. Como Kafka es un sistema distribuido, las colas están particionadas y replicadas sobre múltiples nodos llamados

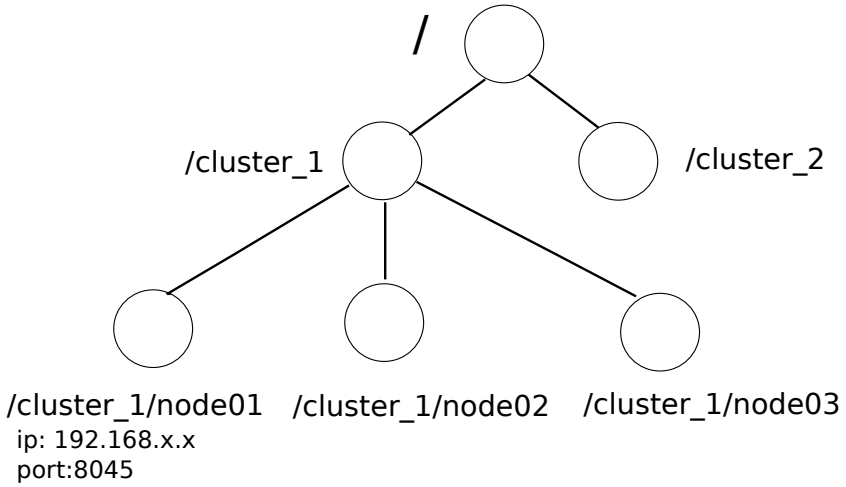


Figura 4.5: Ejemplo Zookeeper

brokers.

Los mensajes son simples arrays de bytes. Esto permite que los desarrolladores almacenen cualquier objeto en cualquier formato.

Lo que hace único a Kafka es que trata cada partición de una cola como un *log* (conjunto ordenado de mensajes). Cada mensaje en una partición tiene asignado un *offset* único. Kafka no monitoriza qué mensajes fueron leídos por cada consumidor. En su lugar, Kafka mantiene todos los mensajes durante una cierta cantidad de tiempo y, son los consumidores los encargados de saber hasta qué punto del *log* leyeron. Esta característica le permite a Kafka soportar un gran número de consumidores y mantener grandes cantidades de datos con muy poca sobrecarga en los sistemas.

4.5. Apache Cassandra

Apache Cassandra [13] es una base de datos distribuida preparada para gestionar grandes cantidades de datos estructurados utilizando clusters formados por recursos heterogéneos. Proporciona alta disponibilidad y es tolerante a fallos.

Cassandra ofrece el potencial que las bases de datos relacionales no pueden proporcionar, como por ejemplo: disponibilidad ininterrumpida, rendimiento escalable de forma lineal, fácil distribución de los datos sobre múltiples centros de datos, etc.

La arquitectura de Cassandra está pensada para tener la capacidad de escalar y ofrecer un servicio continuo. En lugar de utilizar una arquitectura maestro-esclavo, Cassandra utiliza una arquitectura en forma de anillo y sin un nodo que actúe como maestro.

En Cassandra, todos los nodos tienen el mismo papel. No existe el concepto de nodo maestro, todos los nodos se comunican con el resto del mismo modo. Su arquitectura está pensada para escalar sin problema. Esto significa que es capaz de manejar grandes volúmenes de datos y miles de operaciones de entrada/salida por segundo. Además, al no utilizar una arquitectura maestro-esclavo, es tolerante a fallos y puede ofrecer un servicio ininterrumpido.

Capítulo 5

Diseño

En este capítulo se describe la arquitectura del sistema desarrollado con un nivel de abstracción elevado, lo que facilita entender su funcionamiento y percibir como encajan los diferentes subsistemas que la componen.

5.1. Arquitectura del sistema

El sistema desarrollado está inspirado en la arquitectura Lambda [14]. Esta es una arquitectura, sin ninguna implementación asociada, que fue pensada para el desarrollo de sistemas de procesado en tiempo real de grandes volúmenes de datos. La arquitectura Lambda está formada principalmente por tres capas: procesado de datos en tiempo real, procesado de datos históricos y diseminación de resultados. En particular, el sistema desarrollado incluye una nueva capa (adquisición de datos) que permite obtener los datos de la red de sensores del *CiTIUS* y ponerlos a disposición del sistema de procesado. Por lo tanto, y tal y como muestra la Figura 5.1, el sistema se compone de las siguientes cuatro capas:

- **Capa de adquisición**. Ésta es la encargada de recibir los datos generados por los sensores y almacenarlos. El proceso de recepción de los datos es realizado en tiempo real. Los datos son almacenados de forma temporal hasta que la capa de procesamiento de datos en tiempo real pueda consumirlos.
- **Capa de procesamiento de datos históricos**. En esta capa se procesan, de forma iterativa, todos los datos disponibles en el sistema con el fin de

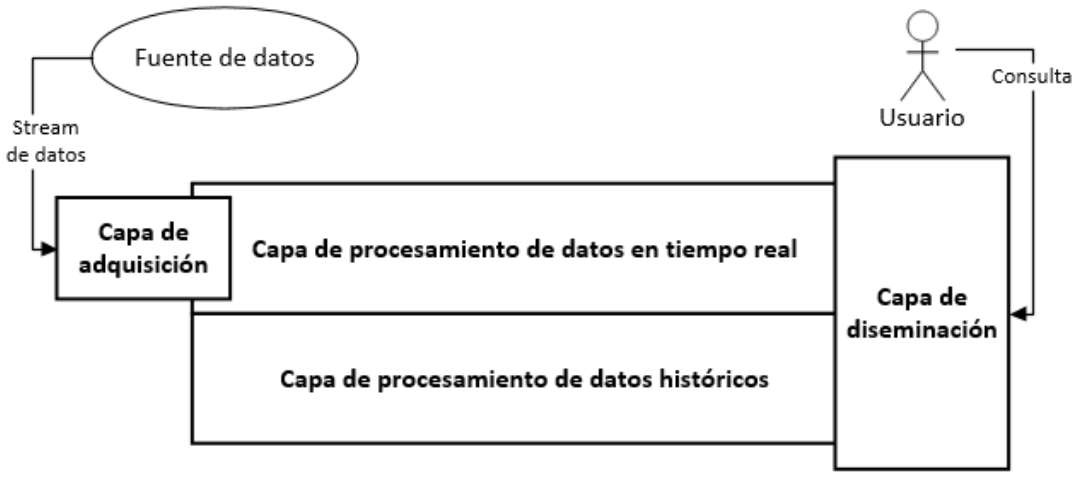


Figura 5.1: Diagrama de arquitectura (nivel de detalle bajo)

generar una vista histórica para cada una de las consultas prediseñadas. El volumen de datos almacenados requerirá de un procesamiento largo e intensivo que podría provocar que los resultados generados no representen el estado actual del sistema, ya que los datos seguirán llegando mientras el procesamiento está en marcha.

- **Capa de procesamiento de datos en tiempo real**. Esta capa nace con el objetivo de gestionar los datos obtenidos mientras los datos históricos son procesados y que, por lo tanto, no podrían estar incluidos en la vista histórica resultante. Esta capa se encarga de recoger los nuevos datos de la capa de adquisición, procesarlos (actualizando la vista en tiempo real) y almacenarlos en un sistema de ficheros distribuido para que puedan ser incorporados en el procesado de datos históricos en las sucesivas iteraciones.

- **Capa de diseminación**. Esta capa se encarga de unificar las vistas correspondientes al procesamiento de los datos históricos y de los datos en tiempo real, con el objetivo de ofrecer al usuario una vista única actualizada del sistema. Esta capa también tendrá un servidor web que alojará la página web donde los usuarios podrán consultar las vistas almacenadas y visualizar los resultados.

Una vez visto el diagrama de la arquitectura y descritas cada una de las capas a alto nivel, en los apartados siguientes se entrará en el diseño de cada una de las capas con mayor detalle. Antes de continuar, en la Figura 5.2 se puede ver la arquitectura con un nivel de detalle mayor que en el diagrama anterior. En él ya se puede ver como están las capas conectadas entre sí y las tecnologías utilizadas en algunas capas.

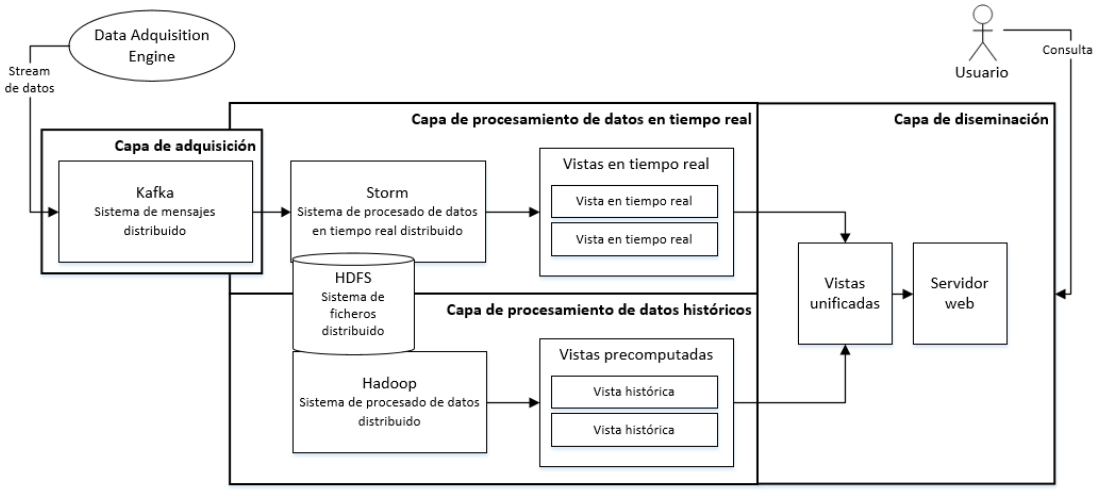


Figura 5.2: Diagrama de arquitectura (nivel de detalle medio)

5.2. Capa de adquisición

El sistema desarrollado en este proyecto no se conecta directamente a los sensores para obtener los datos. Existe una capa intermedia entre los sensores y la capa de adquisición que es totalmente externa y no se tiene ningún control sobre ella. Tiene el nombre de *Data Adquisition Engine* (DAE) y su misión es obtener los datos de los sensores a través de los diferentes protocolos y, posteriormente, enviarlos a una de las colas de Kafka.

Por tanto, esta capa está formada únicamente por el sistema de colas distribuido Kafka. Su misión es servir de almacenamiento temporal para los datos enviados por DAE. Estos datos serán recogidos por la siguiente capa, donde empezará su procesamiento y almacenamiento.

Como el objetivo de esta capa es recuperar la información de la red de sensores, se ha decidido incluir en esta sección el proceso de importación al nuevo sistema de los datos históricos existentes. Esta tarea de migración solo es necesaria ejecutarla una única vez. A partir de ese momento, el sistema desarrollado, a través de la capa de procesado en tiempo real, se encarga de recuperar los nuevos datos.

Actualmente, para el almacenamiento de los datos, se está utilizando una base de datos relacional (PostgreSQL). El plan a seguir para realizar la migración es exportar todos los datos de esa base de datos en ficheros CSV y, a continuación, importar dichos datos en el HDFS.

Para realizar dicha importación se realizó un programa basado en el paradigma MapReduce. La función *map* de dicho programa se encarga de procesar el fichero CSV y preparar la información para ser almacenada en el HDFS. La función *reduce* no se implementó, ya que en este caso no es necesaria.

5.3. Capa de procesamiento de datos en tiempo real

La tecnología principal de esta capa es Storm. Sus tareas son:

1. Recoger los datos que se hayan recibido en la capa de adquisición.
2. Almacenar los datos adquiridos en el HDFS.
3. Procesar los datos adquiridos con el fin de actualizar la vista en tiempo real oportuna.

Una vez establecido el trabajo que debe hacer Storm, el siguiente paso es realizar el diseño de la topología que se va a desplegar en dicha tecnología. La idea inicial se puede ver en la Figura 5.3.

La misión de cada uno de los elementos de esta topología es el siguiente:

- **Obtener datos**. Este *spout* es el encargado de conectarse a la fuente con los datos de los sensores del *CiTIUS* para obtenerlos y transformarlos en las secuencias de tuplas que serán consumidas por los diferentes elementos de la topología.

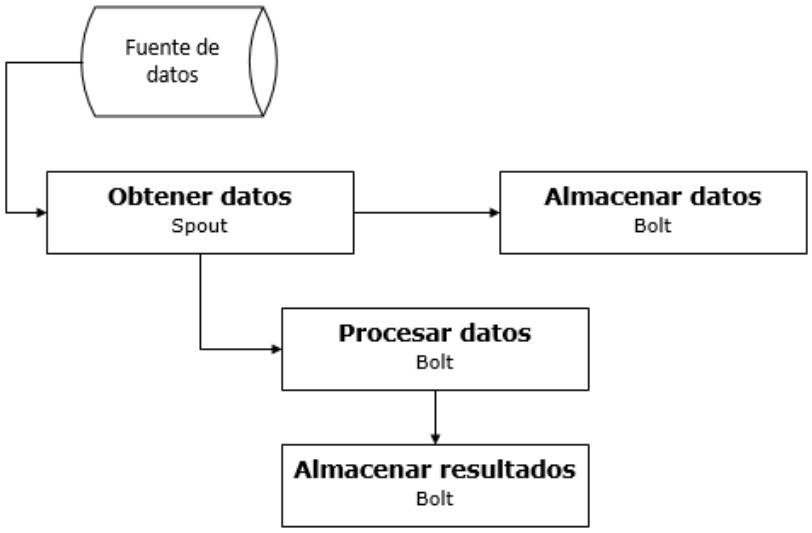


Figura 5.3: Diseño inicial de la topología

- **Almacenar datos**. Este *bolt* se encarga de almacenar los datos en el sistema de ficheros distribuido (HDFS) en su forma original y de forma permanente e inmutable.

- **Procesar datos**. Este *bolt* se encarga de realizar el procesamiento necesario sobre los datos obtenidos.

- **Almacenar resultados**. Este *bolt* se encarga de almacenar el resultado del procesamiento del *bolt* anterior en la vista de tiempo real pertinente.

El siguiente paso es diseñar el diagrama de clases con el fin de implementar posteriormente la topología descrita. En la Figura 5.4 se puede ver el diagrama de clases inicial.

La clase *Topology* es la encargada de construir toda la topología, enlazando los *spouts* y los *bolts* en el orden correcto.

La clase *ObtenerDatos* tiene un único método llamado *siguienteTupla*. Cuando se llama a este método, éste devuelve una tupla que contiene los datos de una medida de un sensor.

Las clases del paquete *bolts* tienen dos métodos:

- *ejecutar*. Aquí es donde se lleva a cabo todo el procesamiento que debe

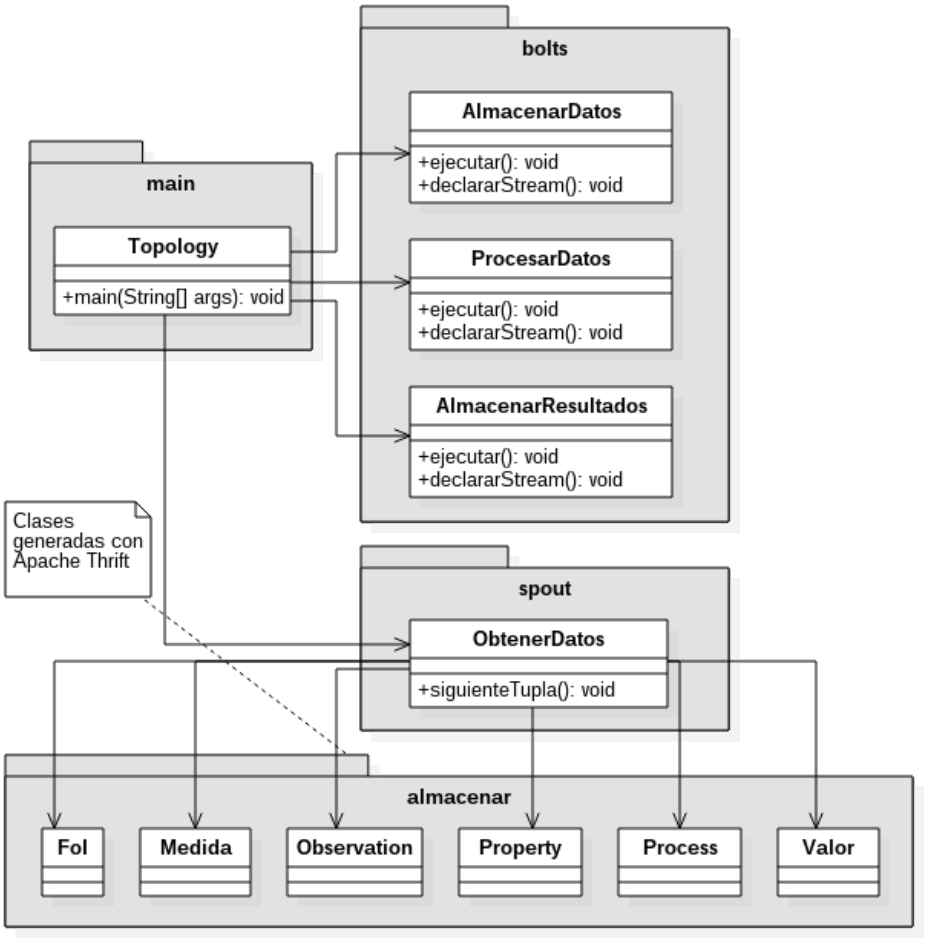


Figura 5.4: Diagrama de clases inicial de la topología

realizar el *bolt* para cumplir con su objetivo.

- *declararStream*. Este método devuelve el tipo de *streams* que emite el *bolt*.

El objetivo de cada una de las clases que pertenecen tanto al paquete *bolts* como al paquete *spouts* es el que se describió anteriormente, cuando se explicó la topología inicial.

Las clases del paquete *almacenar*, al haber sido generadas con Apache Thrift, no es necesario entrar en más detalle.

5.4. Capa de procesamiento de datos históricos

En esta capa se debe diseñar el programa encargado de procesar los datos del histórico y generar la vista pertinente. Además, dicho programa debe estar basado en el paradigma MapReduce.

El diagrama de clases realizado se puede ver en la Figura 5.5.

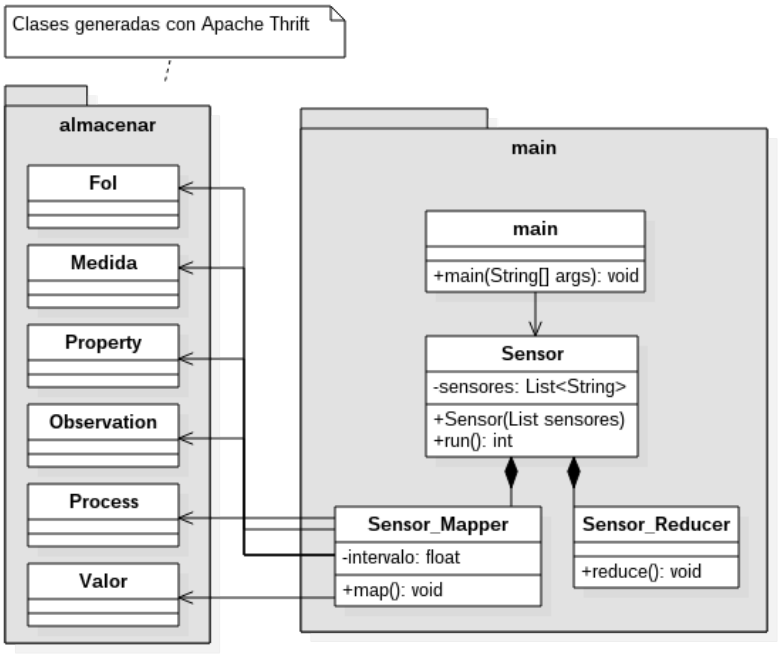


Figura 5.5: Diagrama de clases inicial para el programa de procesado de datos históricos

La clase *Sensor* está formada por dos clases: *Sensor_Mapper* y *Sensor_Reducer*. En la primera clase es donde se implementa la parte de la función *map*. En ella se procesan los datos obtenidos del HDFS y se filtran los datos que deben entrar en el procesado, utilizando la lista con los nombres de los sensores de la clase *Sensor*. Por ejemplo, si se están procesando los datos para generar la vista del consumo del agua, solo se deben coger las medidas realizadas por el sensor de agua, y no por el sensor de gas. En la case *Medida_Reducer* es donde se implementa la parte de la función *reduce*. En ella se agrupan los datos por intervalos de tiempo, por ejemplo de 15 minutos, y se calcula la media de los valores de las medidas que estén en ese intervalo de tiempo. Una vez calculada la media, se actualiza la vista

histórica oportuna con los resultados que se acaban de obtener.

El hecho de agrupar los datos por intervalos tiene como objetivo reducir considerablemente el número de registros que es necesario almacenar en la vista histórica, lo que permite que los datos puedan ser consultados en tiempo real.

5.5. Capa de diseminación

Esta capa tiene tres funciones importantes:

- Gestionar la caducidad de los datos incluidos en la vista de tiempo real.
- Unificar las vistas con el fin de proporcionar al usuario una vista actual del sistema.
- Alojar la página web desde donde los usuarios pueden lanzar consultas.

5.5.1. Gestión de la caducidad de los datos incluidos en la vista de tiempo real

La lógica presente en la capa de procesamiento de tiempo real hacen que esta capa sea mucho más compleja que la capa de procesamiento de datos históricos pero, sin embargo, la arquitectura Lambda permite simplificar su gestión, ya que la vista generada en tiempo real sólo debe de cubrir el intervalo de tiempo no incluido por la vista histórica. Es necesario, por lo tanto, un método que gestione la caducidad de la vista real que indique qué parte de los datos deben de eliminarse. Inspirándonos en [14] se ha desarrollado un sistema de rotación de vistas que permite gestionar la caducidad de los datos. Para cada vista lógica en tiempo real, es necesario mantener 2 vistas físicas que, aunque ambas recibirán datos, sólo una de ellas será consultada. La gestión de la vista física activa se realiza mediante un método de rotaciones dirigido por las iteraciones del procesamiento de datos históricos. Para entender mejor el proceso de rotación de las vistas y la eliminación de los datos obsoletos se va a detallar cómo sería la ejecución de 3 iteraciones.

En la Figura 5.6 se muestra la iteración número 1. Éste sería el punto de partida para la primera vez que se ponga en marcha la arquitectura. Esto quiere

decir que ambas vistas estarían vacías. La iteración comienza con el lanzamiento del programa encargado del procesamiento de datos históricos. Incluso aún cuando no existan datos históricos, la creación y destrucción de tareas MapReduce requiere de un tiempo durante el cual se continuarán recibiendo nuevos datos. Estos nuevos datos son tratados en la capa de procesamiento en tiempo real y los resultados son almacenados en ambas vistas reales. La flecha situada a la derecha de las vistas indica cuál es la vista activa que está siendo utilizada para resolver las consultas realizadas por los usuarios. Es importante destacar que al finalizar esta iteración, la vista de datos históricos está todavía vacía porque inicialmente no existían datos.

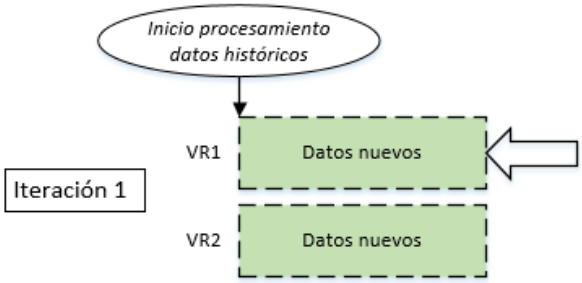


Figura 5.6: Iteración 1

Al inicio de la segunda iteración, la cual puede verse representada en la Figura 5.7, se empiezan a procesar los datos históricos que han llegado hasta ese momento y se rota la vista real activa (en la figura se puede observar que la flecha apunta ahora a VR2) y se eliminan todos los datos contenidos en la vista real desactivada. Los datos que sigan llegando durante el procesado de datos históricos serán, como siempre, incorporados en las dos vistas reales. VR2 contiene durante esta iteración todos los datos que no están presentes en la vista histórica aunque parte de ellos están siendo procesados durante esta iteración.

Al igual que en el caso anterior, la iteración 3, representada en la Figura 5.8, comienza cuando finaliza el procesamiento de los datos históricos de la iteración previa. Nuevamente las vistas reales son rotadas y los datos de la vista desactivada son eliminados. En este punto VR1 es la vista activa y la que contiene los datos no presentes en la vista histórica. Es importante resaltar que en este momento la vista de datos históricos ya contiene el procesado de los datos obtenidos durante

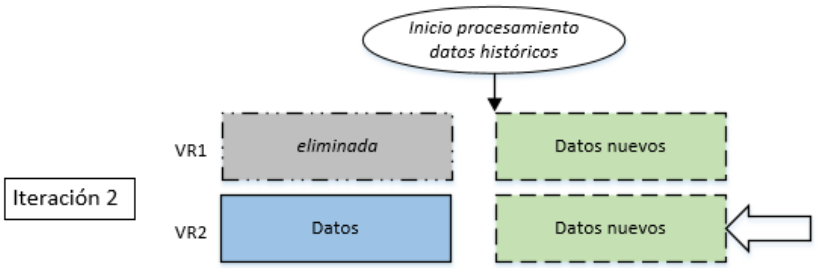


Figura 5.7: Iteración 2

la primera iteración.

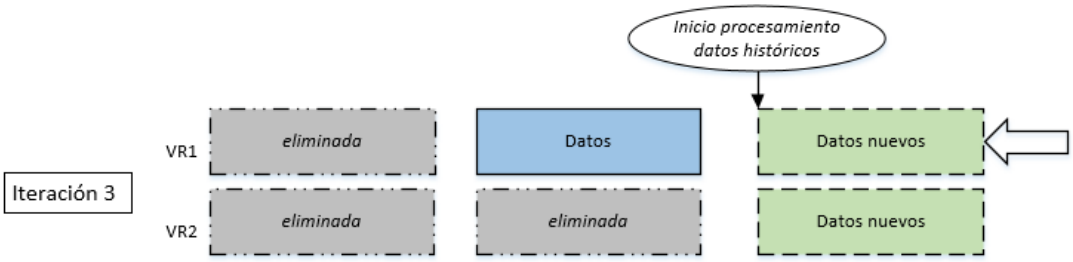


Figura 5.8: Iteración 3

Se puede observar en el ejemplo mostrado que siempre será necesario mantener en la vista real activa los datos obtenidos durante la iteración actual y la anterior. Las iteraciones posteriores seguirán siempre el mismo proceso que el mostrado en el ejemplo.

5.5.2. Unificación de las vistas

Antes de explicar como se va a realizar la unificación primero se debe tener claro qué hay almacenado en cada una de las vistas:

- **Vista histórica**. Aquí los datos están agrupados por un intervalo de tiempo fijo. En este caso se están agrupando en intervalos de 15 minutos.
- **Vista de tiempo real**. Aquí los datos no están agrupados. Es decir, aquí están almacenadas cada una de las medidas recibidas de forma individual.

El primer paso es agrupar de alguna forma los datos de la vista en tiempo real. Aquí hay dos opciones: agruparlos por el mismo intervalo que la vista histórica o agruparlos por un intervalo menor. La ventaja de esta segunda opción es que permitiría un grado de precisión mayor en los datos que son más cercanos a la actualidad. La implementación de estas dos posibilidades es prácticamente igual, ya que la única diferencia es el intervalo de tiempo por el que se agrupan los datos.

Para este proyecto se va a optar por agrupar los datos de la vista de tiempo real por el mismo intervalo que los datos de la vista histórica.

Una vez que se tienen agrupados los datos de la vista en tiempo real solo resta unirlos con los de la vista histórica. El único problema para unirlos es el punto de unión. Se pueden dar dos situaciones que van a ser explicadas a través de un par de ejemplos:

1. En la vista histórica, la agrupación más reciente tiene la siguiente fecha: *12-02-2016 16:00*. Esta agrupación contiene la media de los datos recibidos desde las 16:00 del día indicado hasta las 16:15 de ese mismo día.
   En la vista de tiempo real, el datos más antiguo tiene la siguiente fecha: *12-02-2016 16:11*. Es decir, ese dato (y todos los siguientes) se solapa con el intervalo de la vista histórica.
   Cuando se da esta situación, para obtener un valor para el intervalo conflictivo (de 16:00 a 16:15), se calcula la media entre el valor obtenido en ese intervalo para la vista histórica con el valor obtenido para la vista real.

2. En la vista histórica, la agrupación más reciente tiene la siguiente fecha: *12-02-2016 16:00*. Esta agrupación contiene la media de los datos recibidos desde las 16:00 del día indicado hasta las 16:15 de ese mismo día.
   En la vista de tiempo real, el datos más antiguo tiene la siguiente fecha: *12-02-2016 16:21*.
   En esta situación, no hay intervalo conflictivo, por lo que no es necesario realizar ninguna acción a mayores para unir ambas vistas.

Es importante destacar que el resultado del procesamiento de los datos en tiempo real y su posterior unificación pueden contener pequeñas imprecisiones

que son aceptadas debido a que las vistas en tiempo real son efímeras, y una vez que los datos son incorporados a las vistas histórica, dichas imprecisiones son corregidas.

5.5.3. Página web

Back-End

El *Back-End* tiene dos funciones principales: por un lado procesar las peticiones recibidas de los dispositivos clientes y, por otro lado, llamar a las funciones necesarias para realizar la unión de las vistas tal y como se comentó en el apartado anterior.

Front-End

La aplicación web está formada por dos páginas: la página en donde el usuario rellena los datos necesarios para lanzar la consulta y la página en donde se le muestra el resultado de su consulta.

El *mockup* realizado para la página en donde el usuario rellena los datos de la consulta se puede ver en la Figura 5.9. La información solicitada al usuario es:

- Fecha inicio.
- Fecha fin.
- Intervalo.
- Sensor.
- Despacho. Este desplegable solo estará visible cuando esté seleccionado como sensor *Temperatura despachos*.

El *mockup* realizado para la página en donde se muestra el resultado de la consulta se puede ver en la Figura 5.10. Ésta está formada por dos elementos principales:

- Una gráfica con los resultados.
- La posibilidad de descargar los datos representados en la gráfica en formato CSV.

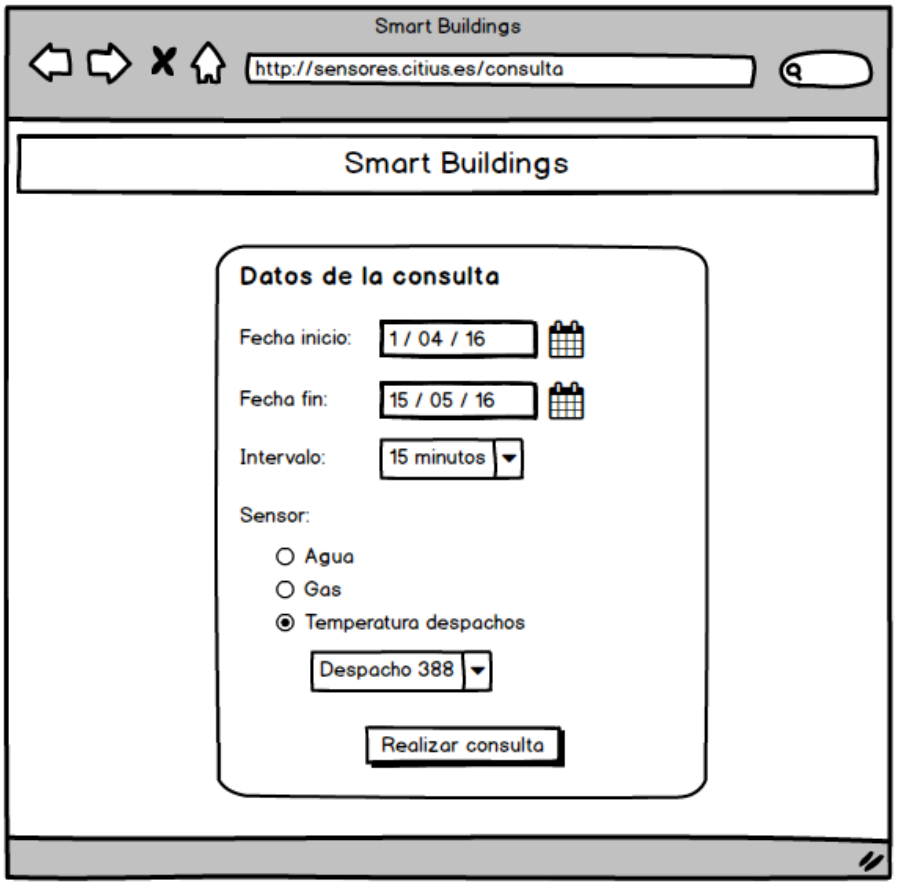


Figura 5.9: Mockup página principal

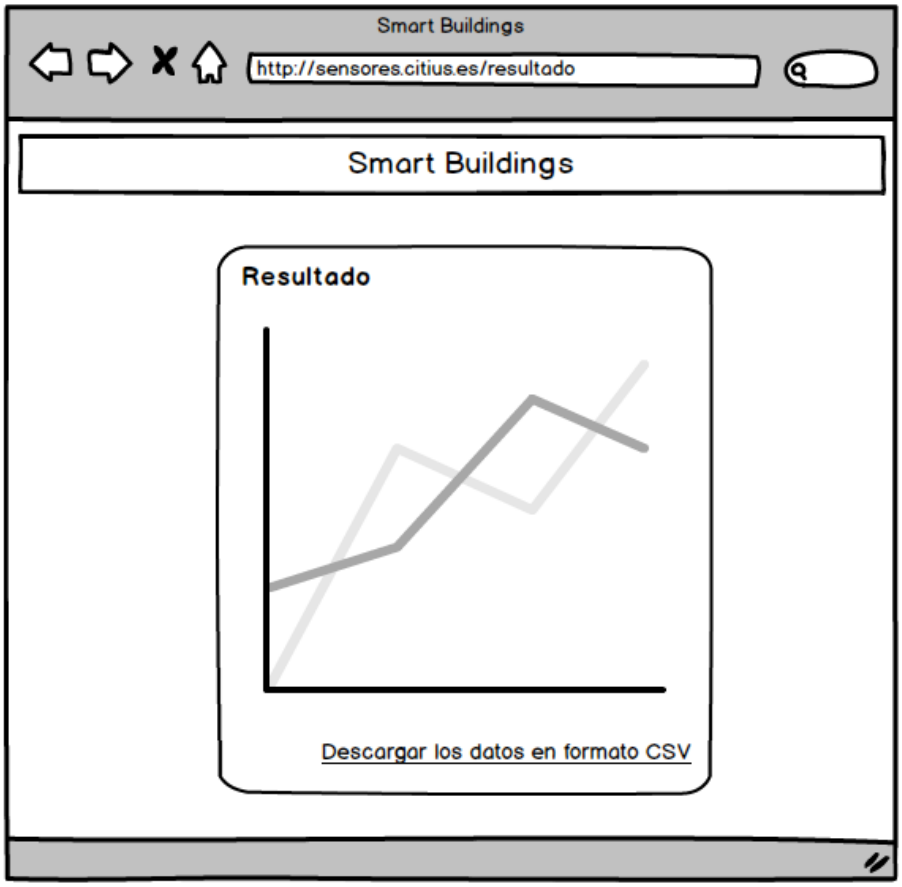


Figura 5.10: Mockup página resultado

Capítulo 6

Implementación

En este capítulo se muestra en detalle como se ha desarrollado cada una de las capas de la arquitectura desarrollada.

En la Figura 6.1 se puede ver el diagrama de arquitectura mostrado en el capítulo anterior pero con un nivel de detalle mayor.

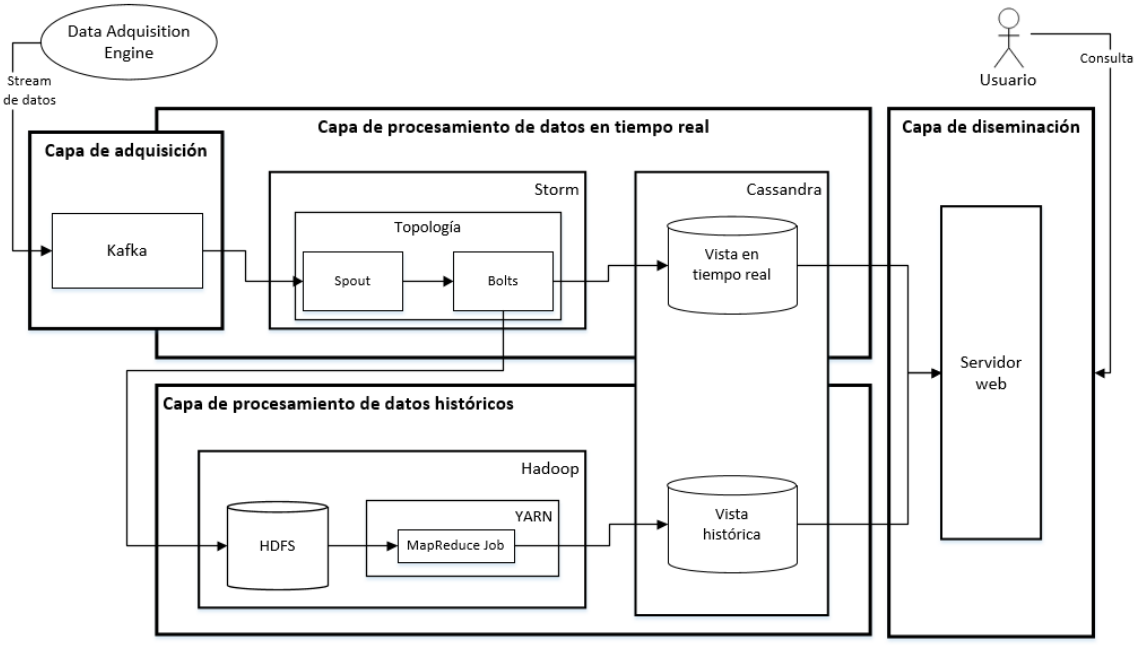


Figura 6.1: Diagrama de arquitectura (nivel de detalle alto)

6.1. Capa de adquisición

Los datos son obtenidos directamente de los sensores a través del DAE empleando diferentes protocolos como BACnet, ModBus, XBee o MOTT. Estos datos luego son enviados a Kafka empleando el formato JSON. Un ejemplo de una medida de un sensor es el siguiente:

```
{
   "medida": {
      "foi": {
         "id": "9",
         "name": "Citius"
      },
      "process": {
         "id": "402",
         "name": "CITIUS_Contador_Agua_Red"
      },
      "observation": {
         "time": "2016-02-28 23:59:56.978",
         "uom": "percent",
         "valor": "100.0",
         "mode": "R",
         "status": "A"
      },
      "property": {
         "id": "40",
         "name": "Agua"
      }
   }
}
```

En este ejemplo se puede ver que el nombre *medida* tiene como valores otros cuatro nombres: *foi, process, observation* y *property*. El nombre *foi* (*Feature Of Interest*) indica en qué edificio está instalado el sensor. El nombre *process* indica el nombre del sensor al que pertenece la medida. El nombre *observation* indica la fecha y la hora de la medida, la unidad de medida, el valor, y el modo y el estado del sensor. Por último, el nombre *property* indica qué está midiendo el sensor.

6.2. Capa de procesamiento de datos en tiempo real

Esta capa debe recoger los datos que haya almacenados en Apache Kafka, procesarlos y almacenarlos en el *Hadoop Distributed File System*. Estas tareas van a ser realizadas por Apache Storm.

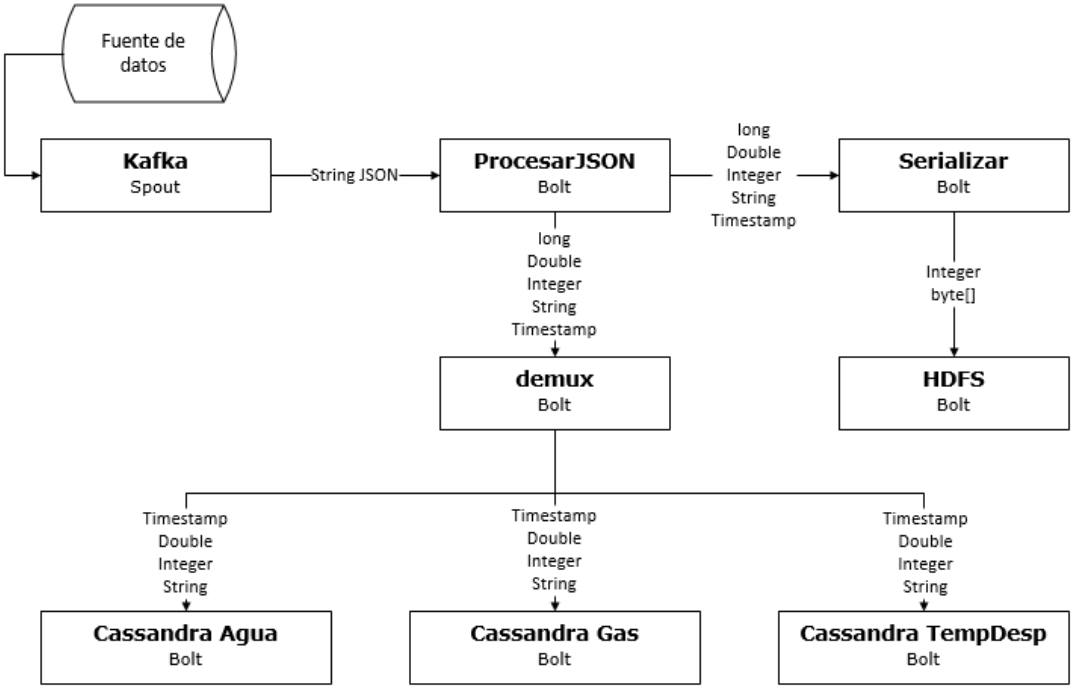


Figura 6.2: Diseño de la topología

Partiendo del diseño mostrado en la Figura 5.3, se ha desarrollado una topología formada por 1 *spout* y 7 *bolts* tal y como se muestra en la Figura 6.2.

La misión de cada uno de los elementos de la topología final es la siguiente:

- ***Kafka Spout***. Este *spout* se encarga de conectarse a Apache Kafka y recoger el siguiente dato que haya sido recibido desde la última conexión. Es importante destacar que solo recoge un dato de cada vez. Una vez obtenido, le envía a *ProcesarJSON Bolt* el contenido del fichero JSON.

- ***ProcesarJSON Bolt***. Este *bolt* se encarga de extraer toda la información del JSON recibido. Una vez extraída, envía, dentro del mismo *stream* de

datos, toda la información obtenida a los siguientes *bolts*: *demux Bolt* y *Serializar Bolt*.

- ***Serializar Bolt***. Este *bolt* se encarga de crear un objeto que contenga toda la información de la medida recibida. A continuación, lo serializa (utilizando Apache Thrift [15]) y lo envía a *HDFS Bolt*.

- ***HDFS Bolt***. Este *bolt* se encarga de almacenar los datos recibidos en el Hadoop Distributed File System. En el HDFS, la información se almacena en *SequenceFiles*. Por tanto, este *bolt* debe crear un *SequenceFile*, meter dentro un determinado número de medidas recibidas, comprimir el *SequenceFile* y, por último, enviarlo al HDFS. Por tanto, este *bolt* actúa como un pequeño *buffer* mientras no llena un *SequenceFile*.

- ***demux Bolt***. Este *bolt* se encarga de determinar de qué sensor proviene la medida. Una vez determinado, se lo envía a uno de los siguientes *bolts*: *Cassandra Bolt - Agua*, *Cassandra Bolt - Gas* o *Cassandra Bolt - TempDesp*. También puede ser que no se le envíe nada a ninguno. Esto es debido a que la medida recibida no es de uno de los sensores de los cuales está creada una consulta prediseñada.

- ***Cassandra Bolt - Agua, Cassandra Bolt - Gas, Cassandra Bolt - TempDesp***. Estos tres *bolts* tienen la misma misión: almacenar la información recibida en Apache Cassandra. La única diferencia es cada uno de ellos debe actualizar una vista distinta en Apache Cassandra. Por ejemplo, *Cassandra Bolt - Agua* deberá actualizar la vista que contiene la información sobre el sensor del agua.

El diagrama de clases final de la topología se puede ver en la Figura 6.3.

Apache Storm proporciona su propia implementación para algunas tareas, como por ejemplo escribir datos en el HDFS. Por eso, *HDFS Bolt* no fue necesario implementarlo. Para crearlo simplemente se instancia un objeto de la clase *SequenceFileBolt* y se le pasan los argumentos necesarios para crear el *bolt* con la configuración deseada. Lo mismo sucede con los *bolts* encargados de escribir en Apache Cassandra y con el *spout* encargado de recuperar los datos de Apache Kafka.

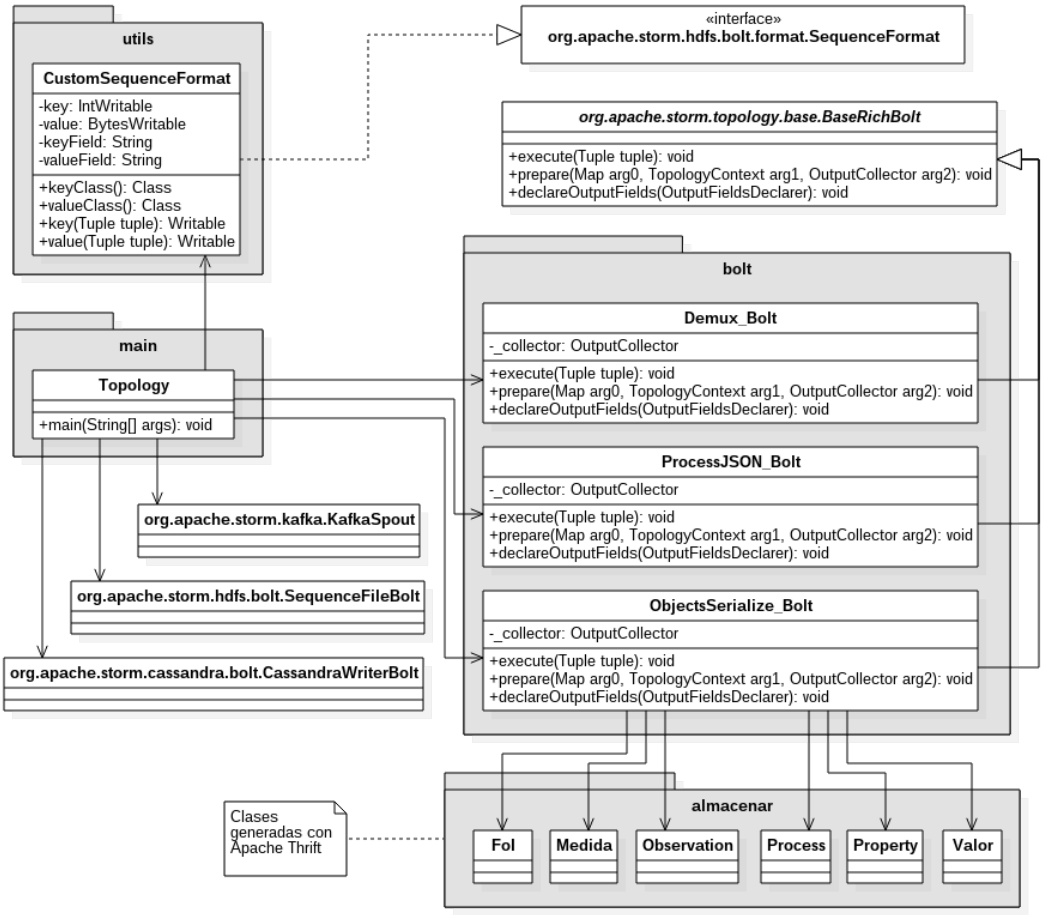


Figura 6.3: Diagrama de clases realizado de la topología

6.3. Capa de procesamiento de datos históricos

En esta capa, el trabajo más importante es desarrollar el programa *MapReduce* para generar las vistas históricas para cada una de las consultas prediseñadas.

El diagrama de clases final asociado a dicho programa se puede ver en la Figura 6.4.

En comparación con el diagrama de clases inicial mostrado en el capítulo anterior, se puede observar que ha entrado en juego alguna clase más. Principalmente, las clases que se han añadido son las que son necesarias extender para poder implementar un programa basado en el paradigma MapReduce. Una clase que sí es importante y que no estaba en el diagrama inicial es la clase *CassandraUtils*. Ésta ha tenido que ser implementada para que la salida de la función

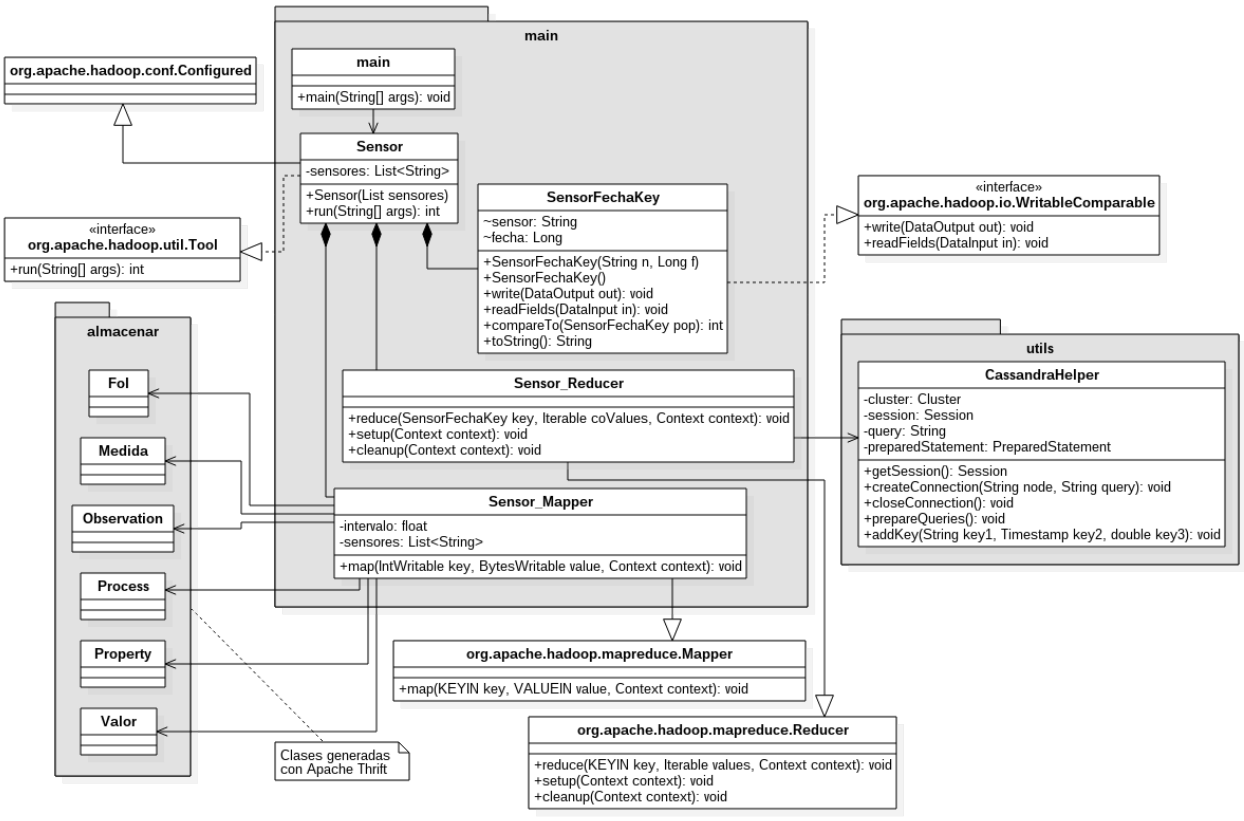


Figura 6.4: Diagrama de clases para el programa de procesado de datos históricos

reduce sea almacenada en Apache Cassandra, es decir, que el resultado de todo el procesamiento realizado sea guardado en la base de datos distribuida.

6.4. Capa de diseminación

Para realizar la parte de la rotación de las vistas se desarrollaron varios *scripts* en *Python* y en *bash*. Los pasos que debían de hacer realizar estos *scripts* son:

1. Cambiar la vista de tiempo real que debe ser consultada.
2. Vaciar la tabla de tiempo real que estaba siendo consultada en la iteración anterior.
3. Mover los datos almacenados por Apache Storm en el HDFS a la carpeta en donde se almacenan los datos históricos.

4. Lanzar los trabajos MapReduce encargados del procesamiento de los datos históricos.

Para realizar la parte de la unificación de las vistas se desarrollaron varias funciones para Apache Cassandra que permiten agrupar los datos por el intervalo deseado. Esto permite que todo el trabajo de agrupar los datos no recaiga directamente sobre la máquina en la que se está ejecutando el servidor web, sino que recaiga sobre todas las máquinas en las que se esté ejecutando Apache Cassandra.

La página web desarrollada es bastante simple, ya que el número de páginas es muy reducido y la complejidad de las mismas tampoco es elevada. Por tanto, se decidió utilizar un *framework* lo más ligero posible que permitiera desarrollar la página web de la forma más cómoda y ágil posible. Uno de los *frameworks* que cumple con estos requisitos es Flask [16].

Flask es un *microframework* para Python. El término *microframework* se refiere a que el núcleo se mantiene lo más simple posible pero que, al mismo tiempo, sea extensible. A diferencia de otros *frameworks*, como Django, Flask no incluye, por defecto, una capa de abstracción para un base de datos o un sistema de validación automática de formularios.

Capítulo 7

Despliegue

En este capítulo se muestra cómo se ha desplegado cada una de las tecnologías sobre un pequeño cluster formado por cuatro máquinas. Los pasos seguidos para la instalación de cada una de las tecnologías se puede ver en el Apéndice A. El contenido de este capítulo se centra en qué servicios se ejecutan en cada una de las máquinas.

Este capítulo se divide en dos apartados. En el primero se indican los servicios que ejecuta el nodo maestro y, en el segundo, los servicios ejecutados por los nodos esclavos. Además, para cada uno de los servicios se incluye una descripción de dicho servicio.

Antes de empezar, en la Figura 7.1 se puede ver el diagrama de componentes realizado antes de empezar con la instalación y configuración de cada una de las tecnologías.

7.1. Nodo maestro

En el nodo maestro se encuentran instalados los siguientes servicios:

- **Storm**. Los servicios de Apache Storm que se lanzan en esta máquina son:
    - *Nimbus*. Se encarga de asignar las tareas a cada uno de los *workers* (es como se le llaman a los nodos esclavo en Storm), monitorizar los nodos, distribuir el código, etc.

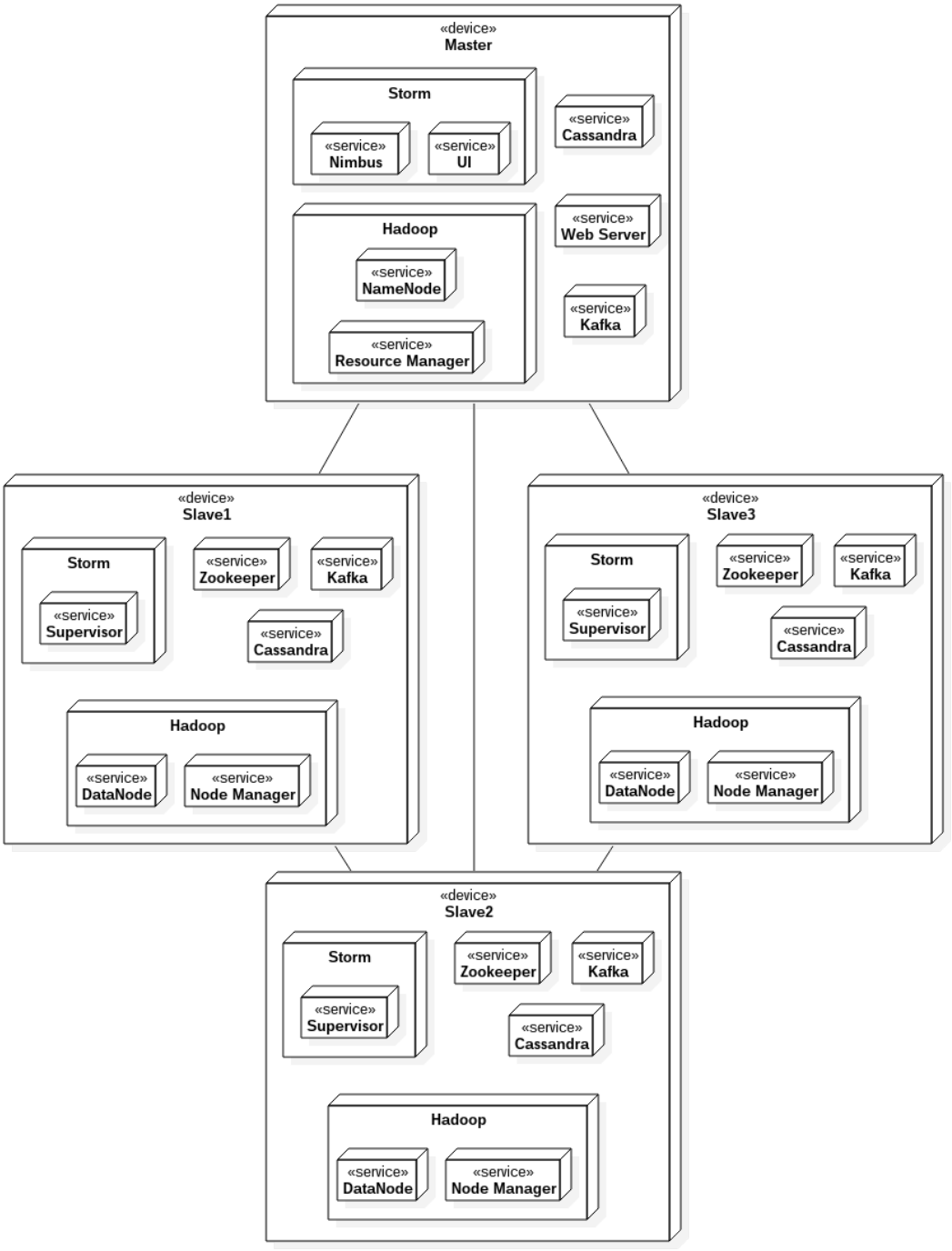


Figura 7.1: Despliegue de la infraestructura sobre 4 máquinas

  - *UI*. Es un servidor web. Accediendo a él se puede ver de forma gráfica el estado del cluster.

- **Hadoop**. Los servicios de Apache Hadoop que son lanzados en esta máquina son:

  - *NameNode*. Es la pieza central del Hadoop Distributed File System. Mantiene un árbol de directorios de todos los ficheros que hay en el sistema de ficheros. Además, debe saber en qué nodo está almacenado cada fichero.
  - *Resource Manager*. Se encarga de gestionar todos los trabajos MapReduce que reciba y asignarlos a los diferentes nodos esclavos.

- **Cassandra**. En Apache Cassandra todos los nodos son iguales, ya que utiliza una arquitectura descentralizada en forma de anillo. Por tanto, a pesar de que en este caso se ejecuta en el nodo maestro, actúa como un nodo más dentro del anillo de Cassandra.

- **Web Server**. Es el servidor web donde está alojada la página web destinada a que los usuarios puedan lanzar consultas.

- **Kafka**. Puede ser lanzado en un cluster de uno o más nodos y cada uno de ellos recibe el nombre de *Broker*. En este caso, se ha decidido lanzar Kafka tanto en los nodos maestros como en los esclavos para mejorar el rendimiento, ya que se prevee una gran carga de datos.

7.2. Nodos esclavos

En los nodos esclavos se encuentran los siguientes servicios:

- **Storm**. El servicio de Apache Storm que se lanza aquí es:

  - *Supervisor*. Se encarga de ejecutar los trabajos enviados por el nodo maestro.

- **Hadoop** Los servicios de Apache Hadoop que se lanzan aquí son:

  - *DataNode*. Se encarga de conectarse al servicio *NameNode* y responde a sus peticiones de operaciones sobre el sistema de ficheros.
  - *Node Manager*. Se encarga de enviar información al *Resource Manager* sobre los recursos disponibles en esta máquina. Además, debe lanzar los trabajos que sean enviados por dicho servicio.

- **Cassandra**. Los nodos esclavos forman parte de la arquitectura en forma de anillo de Cassandra y ejecutan el mismo servicio que el nodo maestro.

- **Kafka**. Los nodos esclavos representan brokers del cluster Kafka y ejecutan el mismo servicio que el nodo maestro.

- **Zookeeper**. Se encargar de mantener información relativa a la configuración de las máquinas. Es utilizado, principalmente, por Apache Storm. Zookeeper solo está instalado en los nodos esclavos. El motivo es que Apache Zookeeper debe ser instalado en un número de máquinas impares. Por tanto, se ha decidido no instarlo en el nodo maestro y solo instalarlo en los esclavos.

Capítulo 8

Pruebas

En este capítulo se describen cada una de las pruebas realizadas sobre la arquitectura desarrollada. Todas las pruebas realizadas fueron de caja negra debido a que realizar pruebas de caja blanca en un sistema distribuido y con tecnologías tan diversas es bastante complejo y requiere de mucho tiempo. Como el tiempo para el desarrollo del proyecto es bastante limitado, se ha optado por realizar solo pruebas de caja negra.

Las pruebas se pueden enforcar de dos formas distintas:

1. Realizar pruebas que se centren en partes pequeñas del sistema pero que sean muy exhaustivas y muy completas. Este tipo de pruebas permiten saber rápidamente qué parte del sistema está fallando. Es decir, permiten aislar el origen del problema con facilidad.

2. Realizar pruebas que abarquen partes grandes del sistema. Este tipo de enfoque permite probar prácticamente la totalidad del sistema con un número reducido de pruebas. El problema de este enfoque es que, si el sistema no pasa una de las pruebas, es muy difícil aislar la parte del sistema que está fallando.

El enfoque utilizado para las pruebas diseñadas va a ser el segundo, es decir, abarcar la mayor parte del sistema con un número reducido de pruebas. El motivo es que se considera que los *frameworks* utilizados son estables y han sido ampliamente probados. Un ejemplo concreto de estas pruebas se puede ver en [17]. Por tanto, el objetivo de este capítulo debe ser probar la integración entre

las diferentes tecnologías.

A continuación se detallan cada una de las pruebas realizadas:

ID	P-1
Descripción	Esta prueba tiene por objetivo probar que el requisito RF-1 es cumplido por el sistema desarrollado. Es decir, la arquitectura es capaz de obtener los datos depositados en la cola de Apache Kafka en tiempo real. Para ello se depositarán en la cola de Kafka varios ficheros JSON que simulen varias medidas de diferentes sensores.
Validación	Una vez que se hayan depositado los diferentes ficheros JSON en Kafka, se accede a uno de los nodos de Cassandra y se comprueba que las vistas de tiempo real tienen los datos que se han enviado a Kafka.
Resultado	Validación correcta

ID	P-2
Descripción	Esta prueba tiene por objetivo probar que el requisito RF-2 es cumplido por el sistema desarrollado. Es decir, que el sistema guarda los datos obtenidos en su forma original. Para ello, primero se depositará en Apache Kafka un fichero JSON simulando la medida de un sensor. A continuación, Apache Storm lo recuperará, lo procesará y lo almacenará en el HDFS. Por último, se desarrolla un pequeño programa MapReduce que reconstruya por completo, a partir de todos los datos almacenados, los ficheros JSON recibidos.
Validación	Si el programa MapReduce reconstruye por completo el JSON enviado inicialmente a la cola se considerará que el sistema pasa la prueba.
Resultado	Validación correcta

ID	P-3
Descripción	Esta prueba tiene por objetivo probar que los requisitos RF-3 y RF-4 son cumplidos por el sistema. Para esta prueba se parte de que el sistema no dispone de ningún dato y que todas las vistas están vacías. El primer paso consiste en depositar en Kafka varios ficheros JSON simulando las medidas de los sensores. A continuación estos datos serán procesados por Storm y generará las vistas reales pertinentes. El siguiente paso es ejecutar el programa encargado de la rotación de las vistas (que incluye el procesamiento de los datos históricos y la generación de las vistas históricas pertinentes). Cuando éste finalice, se depositarán nuevos ficheros JSON simulando nuevas medidas. Éstos datos serán procesados por Storm y generará las vistas reales oportunas.
Validación	Acceder directamente a uno de los nodos de Cassandra y comprobar que en la vista histórica los datos están almacenados por intervalos y que la media de los valores de esos intervalos es correcta con respecto a los datos insertados al principio de la prueba. Una vez comprobada la vista histórica, se accede al contenido de la vista de tiempo real oportuna y se comprueba que los datos depositados después de haber rotado las vistas están almacenados sin agrupar.
Resultado	Validación correcta

ID	P-4
Descripción	Esta prueba tiene por objetivo probar que el requisito RF-5 es cumplido por el sistema. Para ello se accederá a la página web desarrollada y se lanzará una consulta.
Validación	Se descargará de la página web el resultado de la consulta en formato CSV. Una vez descargado, se accederá a uno de los nodos de Cassandra y se comprobará que el contenido del CSV coincide con los datos que están almacenados en cada una de las vistas.
Resultado	Validación correcta

ID	P-5
Descripción	Esta prueba tiene por objetivo probar el requisito RNF-1, tolerancia a fallos hardware. Los nodos en los que se desplegó la arquitectura tienen dos discos duros cada uno. En uno está el SO y en el otro están los datos. Por tanto, para probar la tolerancia a fallos hardware se va a eliminar el disco duro de los datos de uno de los nodos.
Validación	Si toda la arquitectura se sigue ejecutando sin problemas y, ningún dato fue perdido, se considerará que el sistema pasa la prueba.
Resultado	Validación correcta

ID	P-6
Descripción	Esta prueba tiene por objetivo probar el requisito RNF-2. Para ello es necesario desarrollar un programa que simule los datos de los sensores y que genere más de 667 señales cada 10 segundos.
Validación	Se ejecuta el programa de simulación de datos y se comprueba que el sistema es capaz de procesar esa cantidad de datos de entrada. El primer paso para verificarlo es comprobar que todos los servicios se siguen ejecutando y que funcionan de forma correcta. El segundo paso es verificar que en la vista de tiempo real almacenada en Cassandra están llegando todos los datos generados con un retraso menor que 2 segundos.
Resultado	Validación correcta

ID	P-7
Descripción	El objetivo de esta prueba es verificar que se cumple el requisito RNF-3, respuesta en tiempo real. Para ello se realiza un *script* que se encargue de lanzar varias consultas y que mida el tiempo que transcurre entre que envía la petición hasta que obtiene la respuesta a la consulta.
Validación	Si para todas las consultas lanzadas, el tiempo transcurrido entre el lanzamiento de la consulta y la obtención del resultado es menor que 10 segundos, se considerará que el sistema pasa la prueba.
Resultado	Validación correcta

ID	P-8
Descripción	El objetivo de esta prueba es verificar que se cumple el requisito RNF-4, escalabilidad del sistema. Para ello se añadirá una nueva máquina a la infraestructura utilizada.
Validación	Se comprobará, una vez añadida y configurada la nueva máquina, que todos los sistemas funcionan correctamente y detectan a la máquina nueva.
Resultado	Validación correcta

ID	P-9
Descripción	El objetivo de esta prueba es verificar que se cumple con el requisito RNF-7, tolerancia a errores humanos. Para ello se modificará de forma intencionada la forma de calcular la media en los programas MapReduce encargados de generar las vistas históricas. Una vez modificado, se lanzará el programa y éste generará vistas que contienen información incorrecta. Una vez finalizada la ejecución, se corregirá el fallo que se ha introducido de forma intencionada y se volverán a ejecutar los programas MapReduce.
Validación	Se comprobará que las vistas generadas al final de la prueba son correctas. Esto demuestra que un fallo humano en la generación de las vistas no altera los datos almacenados en el HDFS.
Resultado	Validación correcta

Capítulo 9

Conclusiones

En este trabajo de fin de grado se ha desarrollado un sistema escalable y tolerante a fallos enfocado al procesado masivo de datos provenientes de sensores de edificios inteligentes. Dicho sistema debía de gestionar y procesar un gran volumen de datos históricos así como un constante e intenso flujo de datos de entrada y, además, debía de dar soporte a un sistema de consultas en tiempo real sobre todos los datos capturados por los sensores. A través de este desarrollo también se quería probar la idoneidad de emplear la arquitectura Lambda en la construcción de sistemas de procesado de datos provenientes del campo de los *Smart Buildings*, ya que Lambda es una arquitectura sin una implementación asociada, que sólo refleja las capas que debería de tener un sistema genérico de análisis en tiempo real de cantidades ingentes de datos.

El prototipo desarrollado ha demostrado que las tecnologías empleadas para la construcción del sistema fueron adecuadas desde el punto de vista de la arquitectura, ya que cubrieron los requerimientos de cada una de las capas y de los subsistemas asociados a Lambda.

La distribución en capas de la arquitectura permitió emplear un desarrollo iterativo en el que en cada iteración era posible centrarse de forma aislada en un subsistema y en unas tecnologías concretas, lo que facilitó el desarrollo y las pruebas. Las recomendaciones proporcionadas por la arquitectura para la integración de las diferentes capas fueron empleadas satisfactoriamente como puede verse en el Capítulo 5 y en el Capítulo 6. Esta integración permite a los usuarios de forma transparente realizar consultas en tiempo real sobre todos los datos (históricos y

actuales) gestionados por el sistema.

Los *frameworks* de procesamiento empleados (Hadoop y Storm) así como los sistemas de almacenamiento (HDFS y Cassandra) proporcionaron de manera nativa la escalabilidad y tolerancia a errores que se querían para el sistema. Aunque el prototipo resultante fue desplegado sobre un cluster de cuatro nodos (suficiente para procesar los datos generados por el *CiTIUS*), la arquitectura y los *frameworks* empleados permitirían un fácil escalado de la infraestructura en caso de que fuese necesario procesar un mayor volumen de datos.

9.1. Posibles ampliaciones

Una primera ampliación a corto plazo sería la creación de nuevas consultas que permitieran a los usuarios obtener información sobre más elementos del edificio, como el consumo de electricidad, el estado del HVAC o la temperatura en cada uno de los rincones del edificio.

Una segunda ampliación, también a corto plazo, sería la integración de más edificios. En esta prueba de concepto solo se ha utilizado como fuente de datos los sensores del edificio *CiTIUS*. Por tanto, la ampliación consistiría en incorporar más edificios de la Universidad a la arquitectura.

Una tercera ampliación, también a corto plazo, sería el desarrollo de una aplicación para dispositivos móviles. Ésta estaría pensada para que los administradores de los edificios pudieran conocer el estado actual de cada uno de los sistemas más importantes del edificio rápidamente y desde cualquier sitio.

Como posible ampliación a medio-largo plazo estaría el utilizar tecnologías diferentes con el fin de comparar el rendimiento y posibilidades que ofrece cada una de las posibles alternativas. Para este proyecto se ha utilizado Hadoop y Storm porque son los estándares para procesar datos *batch* y en *streaming*, respectivamente. La idea sería utilizar una tecnología que realizara ambos tipos de procesado, con el objetivo de simplificar la arquitectura y reducir la complejidad de mantener dos *frameworks*. Como posibles tecnologías que podrían dar resultado están Apache Spark [18, 19] o Apache Flink (inicialmente conocido como Stratosphere) [20, 21].

Apéndice A

Manual técnico

Este manual explica como instalar cada una de las tecnologías necesarias para desplegar la arquitectura expuesta en este TFG. Como Sistema Operativo se utilizará Ubuntu 14.04 Server de 64 bits.

A.1. Apache Hadoop

A.1.1. Compilando Hadoop

La versión compilada que se puede bajar desde la página oficial de Hadoop no incorpora librerías nativas para la arquitectura de 64 bits. Por tanto, si se usan las librerías que proporcionan, el rendimiento se verá afectado gravemente. Si realmente se quiere obtener el máximo rendimiento es necesario compilar el framework desde el código fuente.

Antes de empezar a compilar hay que instalar las dependencias. Para ello se ejecuta el siguiente comando:

```
root# apt-get install openjdk-7-jdk libprotobuf-dev protobuf-compiler
    maven cmake build-essential pkg-config libssl-dev zlib1g-dev llvm-gcc
    automake autoconf make
```

Una vez instalados los paquetes anteriores, se descarga y descomprime el código fuente:

user$ wget
http://apache.rediris.es/hadoop/common/hadoop−2.7.1/hadoop−2.7.1.tar.gz
user$ tar xvzf hadoop−2.7.1.tar.gz
user$ cd hadoop−2.7.1−src

Una vez llegados hasta aquí, solo queda empezar el proceso de compilación con el siguiente comando:

---

user$ mvn package −Pdist,native −DskipTests −Dtar

Si todo ha funcionado correctamente, la salida del comando anterior debería ser algo así:

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Apache Hadoop Main ................................ SUCCESS [1:11.968s]
[INFO] Apache Hadoop Project POM ......................... SUCCESS [30.393s]
[INFO] Apache Hadoop Annotations ......................... SUCCESS [18.398s]
[INFO] Apache Hadoop Assemblies .......................... SUCCESS [0.246s]
[INFO] Apache Hadoop Project Dist POM .................... SUCCESS [20.372s]
[INFO] Apache Hadoop Maven Plugins ....................... SUCCESS [23.721s]
[INFO] Apache Hadoop MiniKDC ............................. SUCCESS [1:41.836s]
[INFO] Apache Hadoop Auth ................................ SUCCESS [22.303s]
[INFO] Apache Hadoop Auth Examples ....................... SUCCESS [7.052s]
[INFO] Apache Hadoop Common .............................. SUCCESS [2:29.466s]
[INFO] Apache Hadoop NFS ................................. SUCCESS [11.604s]
[INFO] Apache Hadoop Common Project ...................... SUCCESS [0.073s]
[INFO] Apache Hadoop HDFS ................................ SUCCESS [1:30.230s]
[INFO] Apache Hadoop HttpFS .............................. SUCCESS [17.976s]
[INFO] Apache Hadoop HDFS BookKeeper Journal ..... SUCCESS [19.927s]
[INFO] Apache Hadoop HDFS−NFS ............................ SUCCESS [3.304s]
[INFO] Apache Hadoop HDFS Project ........................ SUCCESS [0.032s]
[INFO] hadoop−yarn ....................................... SUCCESS [0.033s]
[INFO] hadoop−yarn−api ................................... SUCCESS [36.284s]
[INFO] hadoop−yarn−common ................................ SUCCESS [33.912s]
[INFO] hadoop−yarn−server ................................ SUCCESS [0.213s]
[INFO] hadoop−yarn−server−common ......................... SUCCESS [8.193s]
[INFO] hadoop−yarn−server−nodemanager .................... SUCCESS [41.181s]
```

```
[INFO] hadoop-yarn-server-web-proxy ..... SUCCESS [2.768s]
[INFO] hadoop-yarn-server-resourcemanager ..... SUCCESS [13.923s]
[INFO] hadoop-yarn-server-tests ..... SUCCESS [0.904s]
[INFO] hadoop-yarn-client ..... SUCCESS [4.363s]
[INFO] hadoop-yarn-applications ..... SUCCESS [0.120s]
[INFO] hadoop-yarn-applications-distributedshell ..... SUCCESS [2.262s]
[INFO] hadoop-yarn-applications-unmanaged-am-launcher .... SUCCESS [1.615s]
[INFO] hadoop-yarn-site ..... SUCCESS [0.086s]
[INFO] hadoop-yarn-project ..... SUCCESS [2.703s]
[INFO] hadoop-mapreduce-client ..... SUCCESS [0.132s]
[INFO] hadoop-mapreduce-client-core ..... SUCCESS [18.951s]
[INFO] hadoop-mapreduce-client-common ..... SUCCESS [14.320s]
[INFO] hadoop-mapreduce-client-shuffle ..... SUCCESS [3.330s]
[INFO] hadoop-mapreduce-client-app ..... SUCCESS [9.664s]
[INFO] hadoop-mapreduce-client-hs ..... SUCCESS [7.678s]
[INFO] hadoop-mapreduce-client-jobclient ..... SUCCESS [9.263s]
[INFO] hadoop-mapreduce-client-hs-plugins ..... SUCCESS [1.549s]
[INFO] Apache Hadoop MapReduce Examples ..... SUCCESS [5.748s]
[INFO] hadoop-mapreduce ..... SUCCESS [2.880s]
[INFO] Apache Hadoop MapReduce Streaming ..... SUCCESS [7.080s]
[INFO] Apache Hadoop Distributed Copy ..... SUCCESS [14.648s]
[INFO] Apache Hadoop Archives ..... SUCCESS [2.602s]
[INFO] Apache Hadoop Rumen ..... SUCCESS [5.706s]
[INFO] Apache Hadoop Gridmix ..... SUCCESS [3.649s]
[INFO] Apache Hadoop Data Join ..... SUCCESS [2.483s]
[INFO] Apache Hadoop Extras ..... SUCCESS [2.678s]
[INFO] Apache Hadoop Pipes ..... SUCCESS [6.359s]
[INFO] Apache Hadoop OpenStack support ..... SUCCESS [5.088s]
[INFO] Apache Hadoop Client ..... SUCCESS [4.534s]
[INFO] Apache Hadoop Mini-Cluster ..... SUCCESS [0.433s]
[INFO] Apache Hadoop Scheduler Load Simulator ..... SUCCESS [7.757s]
[INFO] Apache Hadoop Tools Dist ..... SUCCESS [4.099s]
[INFO] Apache Hadoop Tools ..... SUCCESS [0.428s]
[INFO] Apache Hadoop Distribution ..... SUCCESS [18.045s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14:59.240s
[INFO] Final Memory: 168M/435M
[INFO] ------------------------------------------------------------------------
```

La versión compilada se encuentra en el siguiente directorio:

```
hadoop-2.7.1-src/hadoop-dist/target/hadoop-2.7.1.tar.gz
```

A.1.2. Instalando Hadoop

Se va a explicar como desplegar un nodo master y un nodo slave de Hadoop. Para añadir más nodos de slave simplemente hay que repetir los mismos pasos.

En donde no se indique lo contrario, los comandos se deberán lanzar en cada uno de los nodos, ya sea master o slave.

Lo primero es instalar Java. Aquí hay dos opciones: OpenJRE y Oracle Java. Ambas van a funcionar correctamente. Sin embargo, Apache recomienda utilizar la propia versión de Oracle. Para instalarla se lanzan los siguientes comandos:

```
root# add-apt-repository ppa:webupd8team/java
root# apt-get update
root# apt-get install oracle-java8-installer
```

Para verificar que todo se ha instalado correctamente se puede utilizar:

```
user$ java -version
```

Para tener aisladas todas las actividades relacionadas con Hadoop se va a crear un usuario y un grupo exclusivos para Hadoop. Para ello:

```
root# addgroup hadoop
root# adduser --ingroup hadoop hduser
```

Los servicios de Hadoop utilizan el protocolo SSH para comunicarse entre los distintos nodos. Por tanto, lo primero será instalar el servidor de SSH:

```
root# apt-get install openssh-server
```

Los siguientes comandos se deben lanzar solo en el nodo master hasta que se indique lo contrario.

Se genera la clave SSH con el siguiente comando (hay que ejecutarlo como usuario *hduser*):

hduser@master$ ssh−keygen −t rsa −P ””

Se copia la clave generada al fichero *authorized_keys*:

hduser@master$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys

Se envía la clave generada a todos los nodos slave:

hduser@master$ cat .ssh/id_rsa.pub | ssh hduser@slaveIP 'cat >>
.ssh/authorized_keys'

Sustituyendo *slaveIP* por la IP de cada uno de los nodos slaves.

Los siguientes pasos se deben hacer en todos los nodos, tanto master como slaves.

Ahora se debe modificar el fichero */etc/hosts* para que quede de la siguiente manera:

```
127.0.0.1        localhost

192.168.56.30 HadoopMaster
192.168.56.31 HadoopSlave1
```

Sustituyendo las IPs que están en negrita por las IPs correspondientes al nodo master y al nodo slave.

El siguiente paso es obtener Hadoop, ya sea la versión compilada por nosotros o la versión que se facilita en la página oficial de Hadoop, y colocarla en el directorio */usr/local*. Para descomprimirla se puede ejecutar:

root# cd /usr/local
root# tar xvzf hadoop−2.7.1.tar.gz
root# mv hadoop−2.7.1 hadoop

Una vez descomprimida, se le asigna el propietario y el grupo correcto:

```
root# chown hduser:hadoop -R /usr/local/hadoop
```

Se crean los directorio temporales para el *Namenode* y el *Datanode* y se le asignan el propietario y grupo:

```
root# mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
root# mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
root# chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

Para configurar las variables de entorno se debe modificar el fichero *$HOME/.bashrc* como usuario *hduser*.
Si se están utilizando las librerías precompiladas de Hadoop, es decir, se está utilizando la versión que se ofrece en la página web, se deben añadir las siguientes líneas al fichero indicado anteriormente:

```
# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export
 HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
# -- HADOOP ENVIRONMENT VARIABLES END -- #
```

Sin embargo, si se está utilizando una versión de Hadoop compilada para 64 bits, se debe añadir lo siguiente:

```
# -- HADOOP ENVIRONMENT VARIABLES START -- #
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

```
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_PREFIX=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_PREFIX
export
    HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_PREFIX/lib/native
export HADOOP_CONF_DIR=$HADOOP_PREFIX/etc/hadoop
export HADOOP_HDFS_HOME=$HADOOP_PREFIX
export HADOOP_MAPRED_HOME=$HADOOP_PREFIX
export HADOOP_YARN_HOME=$HADOOP_PREFIX
export
    JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native:$JAVA_LIBRARY_PATH
# -- HADOOP ENVIRONMENT VARIABLES END -- #
```

Los ficheros que se van a modificar a continuación solo es necesario modificarlos en el nodo master. Después, con el comando *rsync* se replicará la configuración a los nodos slave.

Todos los comandos y modificaciones se deben hacer como usuario *hduser*.

En el archivo */usr/local/hadoop/etc/hadoop/hadoop-env.sh* hay que configurar la variable **JAVA_HOME**:

```
JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

En el archivo*/usr/local/hadoop/etc/hadoop/core-site.xml* hay que poner lo siguiente dentro del *tag* **<configuration>**:

```
<property>
   <name>fs.default.name</name>
   <value>hdfs://HadoopMaster:9000</value>
<property>
   <name>dfs.replication</name>
   <value>1</value>
```

```
</property>
<property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
</property>
```

La propiedad con nombre *dfs.replication* indica el número de réplicas que se van a crear de cada dato. El número debe ser menor o igual al número de nodos.

En el archivo */usr/local/hadoop/etc/hadoop/yarn-site.xml* hay que pegar lo siguiente dentro del *tag* **<configuration>**:

```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

En este mismo fichero hay que actualizar las siguientes 3 propiedades:

```
<property>
   <name>yarn.resourcemanager.resource-tracker.address</name>
   <value>HadoopMaster:8025</value>
</property>
<property>
   <name>yarn.resourcemanager.scheduler.address</name>
   <value>HadoopMaster:8035</value>
```

```
</property>
<property>
   <name>yarn.resourcemanager.address</name>
   <value>HadoopMaster:8050</value>
</property>
```

Para configurar el fichero *mapred-site.xml* primero es necesario crearlo a partir de la plantilla:

```
hduser$ cp /usr/local/hadoop/etc/hadoop/mapred−site.xml.template
    /usr/local/hadoop/etc/hadoop/mapred−site.xml
```

Una vez creado, se pega el siguiente contenido en el archivo */usr/local/hadoop/etc/hadoop/mapred-site.xml* dentro del *tag* **<configuration>**:

```
<property>
   <name>mapreduce.job.tracker</name>
   <value>HadoopMaster:5431</value>
</property>
<property>
   <name>mapred.framework.name</name>
   <value>yarn</value>
</property>
```

Ahora se debe crear el fichero */usr/local/hadoop/etc/hadoop/masters*. Aquí es donde se indica cuáles de los nodos son master. Un ejemplo del contenido del fichero es el siguiente:

```
HadoopMaster
```

Hay que hacer lo mismo para los nodos slave. En este caso se configura en el fichero */usr/local/hadoop/etc/hadoop/slaves*. Un ejemplo del contenido sería:

```
HadoopSlave1
```

En este momento ya está todo configurado en el nodo master. Ahora se va a

replicar la configuración al resto de nodos. Para ello se puede utilizar la herramienta *rsync* tal y como se muestra a continuación:

```
hduser@master$ rsync -avxP /usr/local/hadoop/
    hduser@IP:/usr/local/hadoop/
```

Sustituyendo *IP* por la IP de cada uno de los nodos restantes.

Para finalizar con la configuración solo falta formatear el *NameNode*. Para ello:

```
hduser@master$ hdfs namenode -format
```

Para lanzar Hadoop se ejecuta:

```
# hdfs daemons
hduser@master$ start-dfs.sh
# MapReduce daemons
hduser@master$ start-yarn.sh
```

Para parar los demonios se utiliza:

```
# MapReduce daemons
hduser@master$ stop-yarn.sh
# hdfs daemons
hduser@master$ stop-dfs.sh
```

Para verificar que todos los servicios se han lanzado correctamente se puede utilizar el comando *jps*. Si se lanza en un nodo master, la salida debería ser:

```
hduser@master$ jps
    3508 NodeManager
    2838 NameNode
    3207 SecondaryNameNode
    3785 Jps
    3358 ResourceManager
    3007 DataNode
```

Si se lanza en un nodo slave, la salida sería:

```
hduser@slave$ jps
2198 Jps
1930 DataNode
2076 NodeManager
```

Para monitorizar el gestor de recursos de Hadoop se puede acceder a la siguiente URL desde el navegador:

```
http://masterIP:8088
```

Y para acceder a la información sobre el *NameNode* se accede a la siguiente URL:

```
http://masterIP:50070
```

A.2. Apache Kafka

En este apartado se va a mostrar como instalar Apache Kafka en una única máquina.

Lo primero es descargar el paquete binario de Apache Kafka desde su página web oficial. Una vez descargado, para descomprimirlo y colocarlo en una carpeta apropiada para instalarlo se pueden lanzar los siguientes comandos:

```
root# tar -xvf kafka_2.10-0.8.1.tgz
root# mv kafka_2.10-0.8.1 /opt/kafka
root# cd /opt/kafka
```

Para poder lanzar Kafka después es necesario darle permisos de ejecución a los binarios:

```
root# chmod +x /opt/kafka/bin/*
```

Apache Kafka utiliza el servicio Apache Zookeeper[1]. Por tanto, primero es

[1]Apache Zookeeper es un servicio de configuración centralizada y registro de nombres para grandes sistemas distribuidos.

necesario lanzar Zookeeper (viene incluido en el paquete binario de Kafka) con el siguiente comando:

```
user$ /opt/kafka/bin/zookeeper-server-start.sh
    /opt/kafka/config/zookeeper.properties
```

Ahora se va a configurar Kafka. Para ello se deben modificar los siguientes líneas del fichero */opt/kafka/config/server.properties*:

```
listeners =PLAINTEXT://:9092

# The port the socket server listens on
port=9092

# Hostname the broker will bind to. If not set, the server will bind to all
    interfaces
host.name=192.168.56.20

# Hostname the broker will advertise to producers and consumers. If not set,
    it uses the
# value for "host.name" if configured. Otherwise, it will use the value
    returned from
# java.net.InetAddress.getCanonicalHostName().
advertised.host.name=192.168.56.20

# The port to publish to ZooKeeper for clients to use. If this is not set,
# it will publish the same port that the broker binds to.
advertised.port=9092
```

Sustituyendo las IPs que están en negrita por la IP de la máquina donde se esté configurando.

En este momento ya es posible lanzar Kafka con el siguiente comando:

```
user$ /opt/kafka/bin/kafka-server-start.sh
    /opt/kafka/config/server.propeties
```

Con esto ya estarían lanzados tanto Zookeeper, escuchando en el puerto 2181, como Kafka, que estaría escuchando en el puerto 9092.

A.3. Apache Storm

Se va a mostrar como instalar Apache Storm en dos nodos. Uno de los nodos actuará como nodo master y el otro como nodo worker.
En el nodo master correrán los siguientes demonios:

- Zookeeper
- Nimbus
- UI

En el nodo worker correrá el siguiente demonio:

- Supervisor

Lo primero es configurar el fichero */etc/hosts* en todos los nodos. Un ejemplo:

```
127.0.0.1       localhost

192.168.56.10   zkserver1
192.168.56.10   nimbus1
192.168.56.10   stormui1
192.168.56.11   worker1
```

Lo siguiente es instalar Zookeeper. En este caso solo se va a instalar en un nodo, el nodo master. Para instalarlo simplemente se ejecuta:

```
root# apt-get install zookeeperd
```

Antes de continuar, es importante configurar unos parámetros de Zookeeper. En el fichero de configuración */etc/zookeeper/conf/zoo.cfg* se deben modificar las siguientes líneas:

```
autopurge.purgeInterval=24
autopurge.snapRetainCount=3
```

Por defecto, Zookeeper no elimina los datos ni los logs de transacciones antiguas. Con estos dos parámetros se está habilitando la limpieza automática de dicha información.
Una vez configurado, es necesario reiniciar el servicio para aplicar los cambios. Para ello:

```
root# service zookeeper restart
```

A partir de este punto todas las instrucciones deben ser ejecutadas en todos los nodos.

Con el fin de tener aisladas todas las actividades relacionadas con Storm, se va a crear un usuario y un grupo exclusivos. Para crearlos se puede ejecutar:

```
root# adduser --system storm
root# groupadd storm
root# usermod -a -G storm storm
root# chown -R storm:storm /home/storm
root# chmod 775 /home/storm
```

El siguiente paso es descargar Apache Storm de su página oficial y moverlo a la carpeta */home/storm*. Para descomprimirlo y asignarle los permisos apropiados:

```
root# cd /home/storm
root# tar xvzf apache-storm-1.0.0.tar.gz
root# chown -R storm:storm /home/storm/apache-storm-1.0.0
root# chmod 775 /home/storm/apache-storm-1.0.0
```

Para configurar Storm es necesario editar el fichero */home/storm/apache-storm-1.0.0/conf/storm.yaml* y editar las siguientes líneas:

```
storm.zookeeper.servers:
- "zkserver1"
nimbus.seeds: ["nimbus1"]
```

En las dos primeras líneas se indican los servidores de *Zookeeper*. La última línea indica en qué máquina estará corriendo el servicio *Nimbus*[2] de Apache Storm.

Si para lanzar una topología es necesario utilizar librerías de terceros hay que configurar el parámetro *'java.library.path'*. Por defecto, este parámetro tiene el siguiente contenido:

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

Si, por ejemplo, la topología necesita las librerías nativas de Apache Hadoop, entonces es necesario añadir la siguiente línea al fichero de configuración mencionado anteriormente:

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

En este momento ya estaría todo listo para ejecutarse. En el nodo master se lanzarían los servicios *Nimbus* y *UI*. Para ello:

```
root# /home/storm/apache-storm-1.0.0/bin/storm nimbus &
root# /home/storm/apache-storm-1.0.0/bin/storm ui &
```

Ahora faltaría lanzar el servicio *Supervisor* en cada uno de los nodos que actúen como slaves. Para ello se ejecuta el siguiente comando en cada uno de ellos:

```
root# /home/storm/apache-storm-1.0.0/bin/storm supervisor &
```

Para acceder a la *UI* se puede acceder a la siguiente URL desde un navegador:

```
http://nimbus1:8080
```

[2]El servicio *Nimbus* es el que se encarga de realizar la asignación y monitorización de las tareas en las distintas máquinas.

A.4. Apache Cassandra

Se va a mostrar como instalar Apache Cassandra en dos nodos. Ambos nodos se configuran del mismo modo a excepción de que uno de ellos actuará como nodo semilla.

Para empezar es necesario descargar Apache Cassandra desde la página web oficial y moverlo a la carpeta */opt*. Una vez hecho esto, se puede descomprimir ejecutando:

```
root# cd /opt
root# tar xvzf apache-cassandra-3.5-bin.tar.gz
```

El siguiente paso es editar el fichero de configuración */opt/apache-cassandra-3.5/conf/cassandra.yaml*. Las líneas que hay que modificar son las siguientes:

```
cluster_name: 'Cluster1'
seeds: "192.168.56.16"
listen_address: 192.168.56.16
rpc_address: 192.168.56.16
```

La primera línea define el nombre del cluster. La segunda línea indica la IP de los nodos que actúan como nodos semilla. La tercera y cuarta línea indica la IP del nodo que se está configurando, es decir, es la IP del propio nodo.

Una vez configurado, para lanzar Cassandra se ejecuta el siguiente comando (automáticamente se lanza en *background*):

```
user$ /opt/apache-cassandra-3.5/bin/cassandra
```

Una vez que se haya lanzado en todos los nodos, se puede comprobar el estado del cluster completo con el siguiente comando:

```
user$ /opt/apache-cassandra-3.5/bin/nodetool status
```

Para lanzar la consola interactiva de Cassandra se utiliza el siguiente comando:

```
user$ /opt/apache-cassandra-3.5/bin/cqlsh ip_nodo
```

Es necesario indicarle al comando uno de los nodos del cluster. No es necesario que sea el nodo semilla. Y no importa cuál, ya que todos deberían ver la misma información.

El trabajar con el tipo de dato *timestamp* para la gestión de la fecha y hora, incluye, en Cassandra, trabajar también con las zonas horarias. Para añadir el soporte de las diferentes zonas horarias es necesario instalar el módulo de Python llamado *pytz*. Para instalarlo:

```
root# pip install pytz
```

Una vez instalado solo queda configurar la zona horaria que se desea utilizar. Para ello es necesario establecer la variable de entorno *TZ* en la terminal donde se vaya a lanzar la consola interactiva de Cassandra. Un ejemplo de como definir esta variable es:

```
export TZ="Europe/Madrid"
```

Si ahora se lanzara la consola de Cassandra, todas las fechas y horas se mostrarán con la zona horaria que se acaba de definir. Si no se definiera dicha variable, todas las fechas y horas se mostrarían en UTC.

Apéndice B

Manual de usuario

La página de inicio puede verse en la Figura B.1. En ella es donde se solicitan los datos necesarios para realizar la consulta. Estos datos son:

- Fecha inicio
- Fecha fin
- Intervalo
- Tipo de sensor

Una vez rellenados los datos anteriores se puede lanzar la consulta pulsando el botón *Realizar consulta*. En el la Figura B.2 se puede ver un ejemplo de cómo se rellenarían los datos. En esa figura se puede ver que al seleccionar como *Tipo de sensor Temperatura despachos* aparece un nuevo desplegable que permite elegir sobre qué despacho se quiere realizar la consulta.

Una vez lanzada la consulta, se es redirigido a la página web en donde se muestran los resultados. Un ejemplo de dicha página se puede ver en la Figura B.3. Como se puede observar, en la gráfica se muestra el resultado de la consulta de forma resumida. Para obtener todos los datos se ofrece la posibilidad de descargarlos pinchando en el link situado debajo de la gráfica.

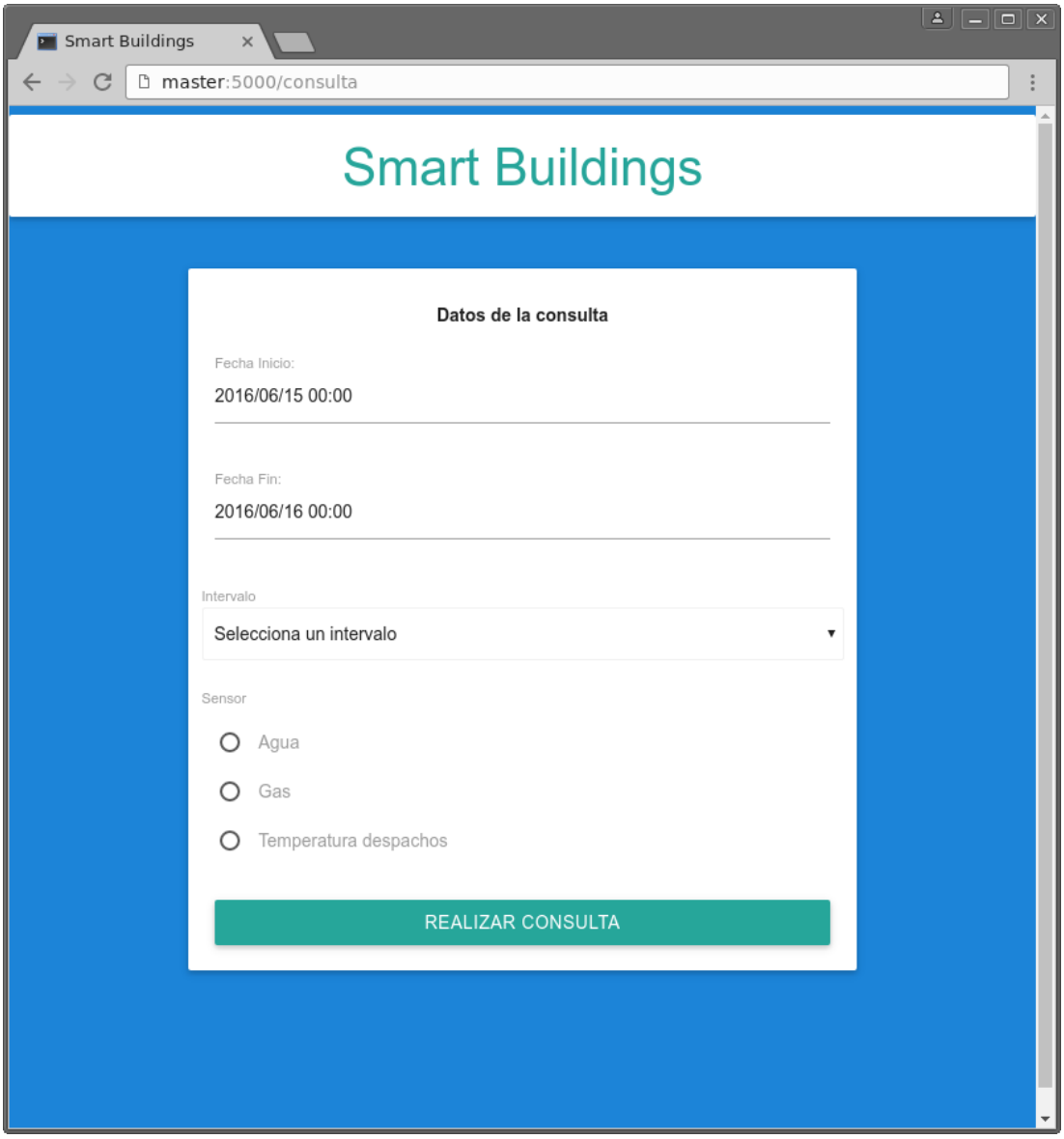


Figura B.1: Página web - Realizar consulta

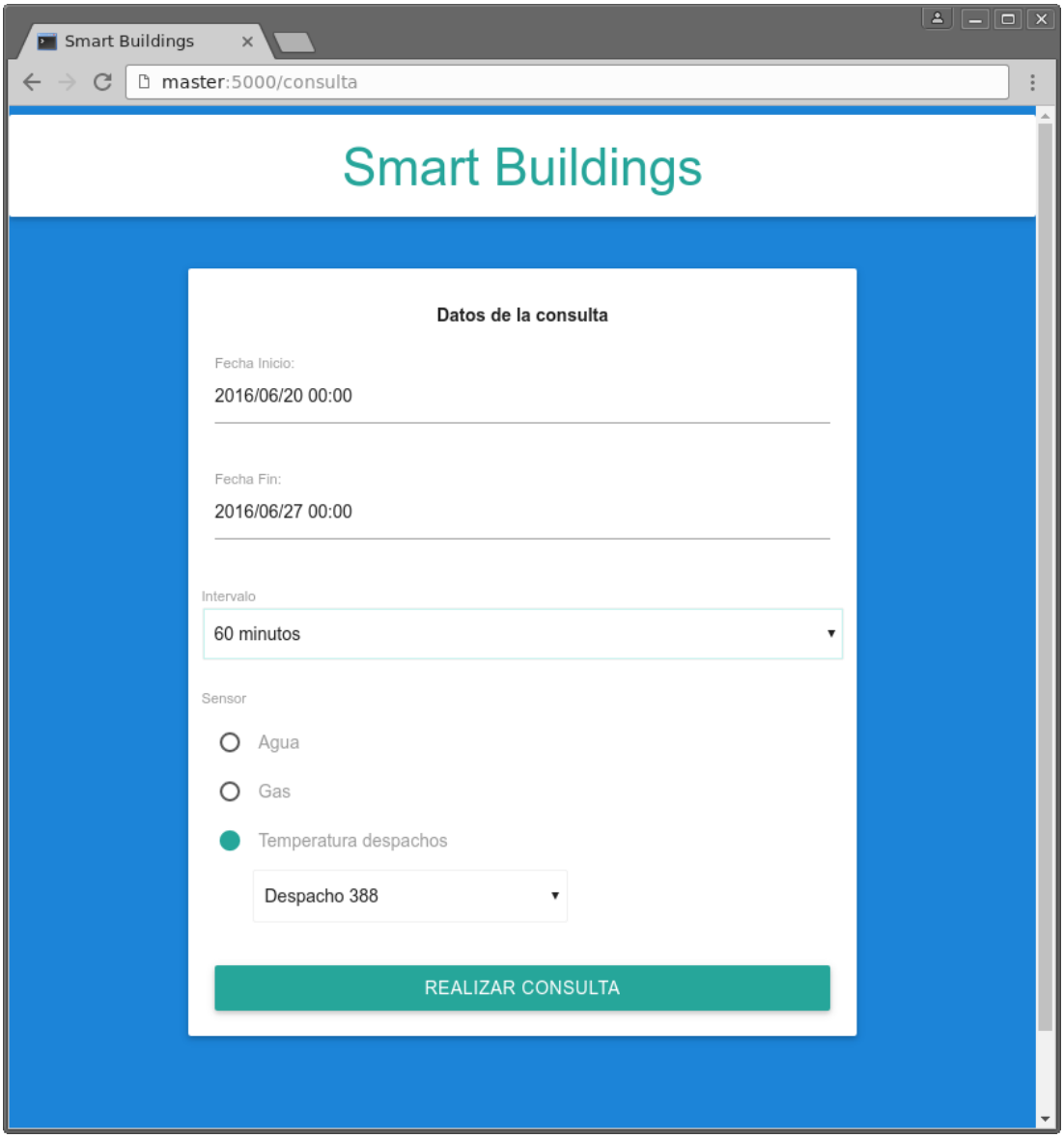


Figura B.2: Página web - Realizar consulta

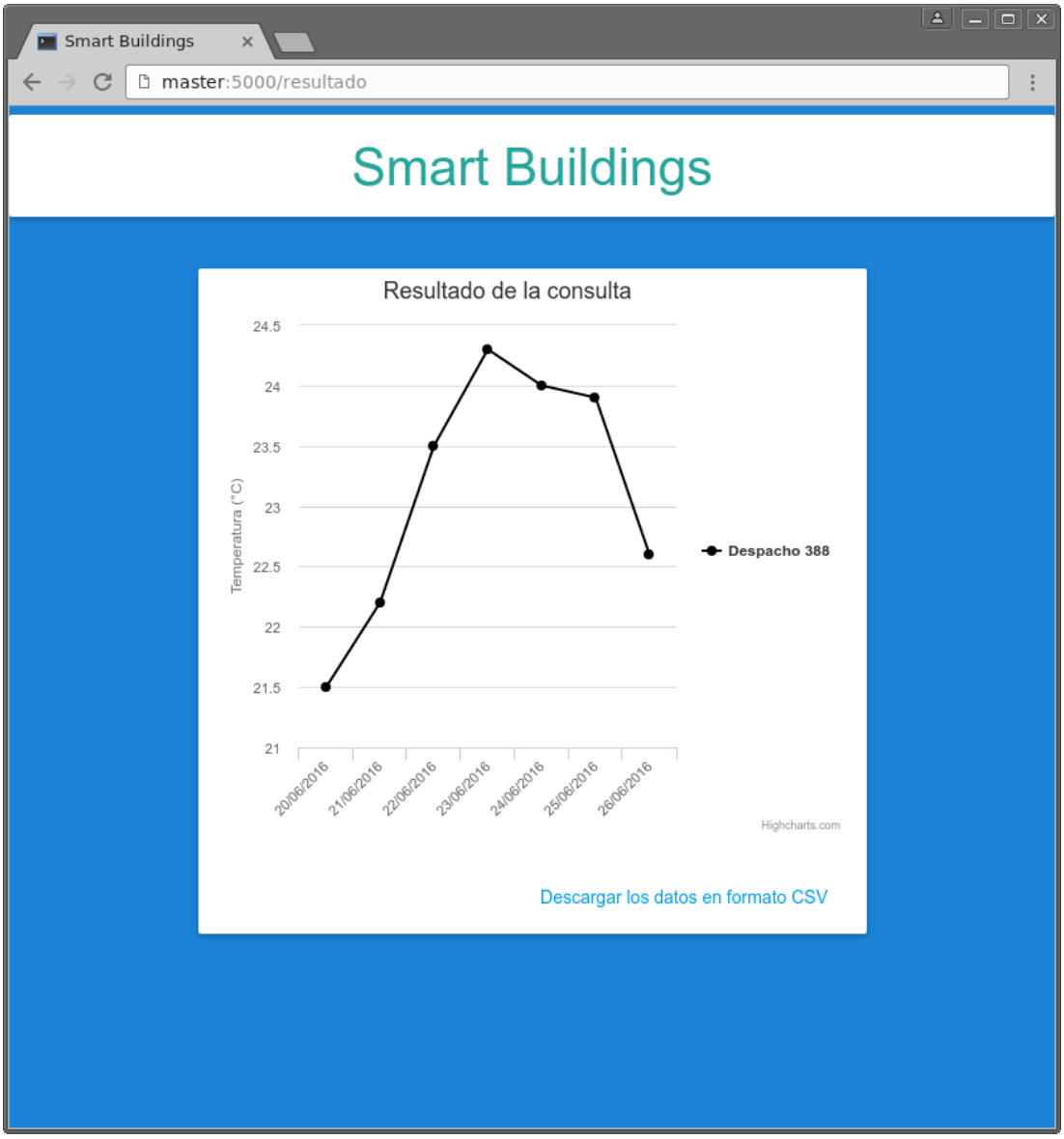


Figura B.3: Página web - Resultados

Bibliografía

[1] A Guide. Project management body of knowledge (pmbok® guide). In *Project Management Institute*, 2001.

[2] Overleaf. http://www.overleaf.com/, (Accedido en julio 2016).

[3] GitLab CiTIUS. https://gitlab.citius.usc.es/, (Accedido en julio 2016).

[4] Vitae Consultores. Guía Salarial Sector TI Galicia 2015-2016. http://vitaedigital.com/download/NTY2/Guia_Salarial_Sector_TI_Galicia_2015-2016.pdf, (Accedido en junio 2016).

[5] Universidade de Santiago de Compostela. Gestión económica de I+D. http://imaisd.usc.es/ftp/oit/documentos/591_gl.pdf, (Accedido en julio 2016).

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] Apache Hadoop. http://hadoop.apache.org/, (Accedido en junio 2016).

[8] Apache Storm. http://storm.apache.org/, (Accedido en junio 2016).

[9] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[10] Apache ZooKeeper. http://zookeeper.apache.org/, (Accedido en junio 2016).

[11] Apache Kafka. http://kafka.apache.org/, (Accedido en junio 2016).

[12] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[13] Apache Cassandra. http://cassandra.apache.org/, (Accedido en junio 2016).

[14] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[15] Apache Thrift. http://thrift.apache.org/, (Accedido en junio 2016).

[16] Flask. http://flask.pocoo.org/, (Accedido en julio 2016).

[17] David Mera, Michal Batko, and Pavel Zezula. Speeding up the multimedia feature extraction: a comparative study on the big data approach. *Multimedia Tools and Applications*, pages 1–21, 2016.

[18] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[19] Apache Spark. http://spark.apache.org/, (Accedido en junio 2016).

[20] Apache Flink. http://flink.apache.org/, (Accedido en junio 2016).

[21] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, 2014.