



Universidad de Granada

BIG DATA EN LA PREDICCIÓN DEL TRÁFICO: OPTIMIZACIÓN Y DIFUSIÓN EN LA NUBE

Presentada por

ANTONIO J. FERNÁNDEZ ARES

Como defensa para
**MÁSTER EN INGENIERÍA
DE COMPUTADORES Y REDES**

Directores

PEDRO A. CASTILLO VALDIVIESO
ANTONIO M. MORA GARCÍA

Firmado: Antonio J. Fernández Ares

Septiembre 2014

Antonio J. Fernández Ares: *Big Data en la predicción del tráfico: Optimización y difusión en la Nube*, Proyecto Fin de Máster, © Creative Commons Attribution-NonCommercial-NoDerivs 3.0 License
Septiembre de 2014

VISTO BUENO

Los Prof. Dr. D. Pedro A. Castillo Valdivieso y Antonio M. Mora García del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada,

CERTIFICAN:

Que la memoria titulada:

"Big Data en la predicción del tráfico: Optimización y difusión en la Nube"

ha sido realizada por **D. Antonio J. Fernández Ares** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de **Máster en Ingeniería de Computadores y Redes**.

En Granada, a 1 de Septiembre de 2014.

Los Directores del proyecto Fin de Máster:

Fdo. Pedro A. Castillo Valdivieso y Antonio M. Mora García

DECLARACIÓN

El estudiante *Antonio J. Fernández Ares* y los directores del Proyecto Fin de Máster *Pedro A. Castillo Valdivieso* y *Antonio M. Mora García* garantizamos, al firmar esta memoria, que el trabajo ha sido realizado por el estudiante bajo la dirección de los directores del Proyecto Fin de Máster y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Septiembre de 2014

Antonio J.
Fernández Ares

Pedro A. Castillo
Valdivieso

Antonio M. Mora
García

A mis pies por apoyarme en todo momento
y a mis brazos que siempre han estado a mi lado.

RESUMEN

Dados los beneficios de un sistema de información sobre el estado del tráfico y de predicción del uso de la red viaria por parte de los vehículos, se plantea el desarrollo de un sistema de información de bajo coste y autónomo para monitorizar el tráfico y conocer el estado de las carreteras en tiempo real.

Este trabajo se enmarca en el proyecto SIPEsCA, cuyos objetivos son disponer de información acerca de los flujos de tráfico que se producen entre ciudades, para facilitar la gestión de los desplazamientos por parte de los ciudadanos.

En este trabajo se recoge la parte de optimización del procesamiento de los datos, los mecanismos de predicción y la publicación en la nube, realizados por el investigador contratado y estudiante del máster que defiende este Proyecto Fin de Máster.

ABSTRACT

Given the benefits of an information system of prediction of the traffic and the use of the roads by vehicles. The development of an information system and autonomous low cost to monitor the traffic and know the road conditions in real time is suggested.

This work is a part of the SIPEsCA project, which aims to provide information about traffic flow that occur between cities to facilitate the management of travel by citizens.

In this work, is presented the processing of data, the prediction mechanisms and the cloud data publication. This work has been done by the contracted researcher and masters student defending this Master Final Project.

PUBLICACIONES CIENTÍFICAS IMPLICADAS

Hasta la fecha de realización de esta memoria se han realizado las siguientes publicaciones en congresos y revistas internacionales en las que el estudiante como personal investigador del proyecto ha participado:

- *Sistema de información autónomo y de bajo coste para conocer el estado de las carreteras en tiempo real*
JCE 2012. III Jornadas de Computación Empotrada. Elche, Alicante. Septiembre 2012
<http://atccongresos.ugr.es/jce/>
- *Estudio de los indicadores de exposición en el área metropolitana de Granada mediante un nuevo sistema de información de bajo coste*
Congreso Internacional de Seguridad Vial. Mayo 2013
<http://www.ispcs.es/seguridadvial/>
- *The L-Co-R co-evolutionary algorithm: a comparative analysis in medium-term time-series forecasting problems*
ECTA-IJCCI 2013. International Conference on Evolutionary Computation Theory and Applications. Septiembre 2013
<http://www.ecta.ijcci.org/>
- *Desarrollo de servicios para una Arquitectura Orientada a Servicios para Algoritmos Evolutivos*
MAEB - CEDI 2013. Septiembre 2013
<http://bioinspired.dacya.ucm.es/maeb2013/>
- *Predicción de tráfico mediante co-evolución de Redes Neuronales de Funciones de Base Radial y selección de variables de entrada*
MAEB - CEDI 2013. Septiembre 2013
<http://bioinspired.dacya.ucm.es/maeb2013/>
- *Estudio de los indicadores de exposición al riesgo mediante un sistema de monitorización del tráfico basado en la tecnología Bluetooth*
JCE - CEDI 2013. IV Jornadas de Computación Empotrada. Septiembre 2013
<http://atccongresos.ugr.es/jce/>
- *Using SOAP and REST web services as communication protocol for distributed evolutionary computation*

International Journal Of Computers & Technology, ISSN
22773061. 2013

En mayo de 2014 se recibió la notificación de aceptación definitiva de un capítulo de libro en publicación internacional describiendo las tecnologías base del proyecto SIPESCA:

- *Studying individualized transit indicators using a new low-cost information system* P.A. Castillo, A. Fernández-Ares, P. García-Fernández, P. García-Sánchez, M.G. Arenas, A.M. Mora, G. Romero, V. Rivas, J.J. Merelo Book: Industry and Research Perspectives on Embedded System Design. 2014
<https://sites.google.com/a/softteam-rd.eu/book/home>

En abril de 2014 se publicaron y presentaron los siguientes trabajos en el congreso internacional EvoStar 2014 y en sendas revistas internacionales:

- *Co-Evolutionary Optimization of Autonomous Agents in a Real-Time Strategy Game* Congreso Internacional EvoStar 2014
<http://www.evostar.org/>
- *Tree depth influence in Genetic Programming for generation of competitive agents for RTS games* Congreso Internacional EvoStar 2014
<http://www.evostar.org/>
- *An object-oriented library in JavaScript to build modular and flexible cross-platform evolutionary algorithms* Congreso Internacional EvoStar 2014
<http://www.evostar.org/>
- *Nuevo sistema de información de bajo coste basado en tecnología bluetooth para conocer el estado de las carreteras en tiempo real* Sociedad Mexicana De Inteligencia Artificial - Revista Komputer Sapiens. Volumen Especial en Programación de Tareas - Scheduling e Inteligencia Artificial: Nuevos Retos
<http://www.komputersapiens.org>
- *Monitorización del estado de las carreteras mediante un sistema autónomo y de bajo coste basado en la tecnología Bluetooth* SECURITAS VIALIS, Journal Europeo de Tráfico, Transporte y Seguridad Vial. Editorial Springer

A final de abril de 2014 se comunicó la aceptación de un trabajo de investigación en el congreso International work-conference on Time Series (ITISE 2014). El trabajo se presentó el 26 de junio.
<http://itise.ugr.es/>

- *Traffic flow forecasting*

Y a principio de mayo, se comunicó la aceptación de un trabajo de investigación en el congreso international 8th International Symposium on Intelligent Distributed Computing - IDC'2014

- *A methodology to develop Service Oriented Evolutionary Algorithms*

y de un trabajo en la revista nacional ReVisión (número de mayo de 2014):

- *Progamer: aprendiendo a programar usando videojuegos como metáfora para visualización de código*

A principio de junio de 2014 se presentó un trabajo en el I Congreso de la Sociedad Española para las Ciencias del Videojuego, CoSECiVi 2014 (Barcelona, 24 de junio de 2014)

<http://gaia.fdi.ucm.es/sites/cosecivi14/es/>

- *Designing Competitive Bots for a Real Time Strategy Game using Genetic Programming*

Y se comunicó la aceptación de un trabajo en las V Jornadas de Computación Empotrada (Valladolid, 17 al 19 de septiembre de 2014) <http://atccongresos.ugr.es/jce/>

- *Studying individualized transit indicators using a new low-cost information system*

Además, un trabajo enviado a una revista internacional recibió la comunicación de "major revision" por lo que se encuentra en segunda revisión:

- *Population size adaptation in distributed evolutionary algorithms on heterogeneous clusters* Revista internacional "Applied Soft Computing"

Actualmente está en redacción un trabajo sobre predicción de series temporales para enviar a revista internacional.

Además de las publicaciones científicas, durante el desarrollo del proyecto, se han llevado a cabo y presentado dos proyectos fin de carrera de Ingeniería Informática e Ingeniería Electrónica estrechamente relacionados con los temas tratados en SIPESCA:

- *Aplicación de algoritmos de data mining al procesamiento de datos de movilidad* realizado por el estudiante **Jonathan Ramírez Taboada** Ingeniería Informática. Curso 2012-2013.
- *Reconocimiento de dispositivos Bluetooth en tiempo real. Aplicaciones* realizado por el estudiante **Óscar Montero Soto**. Ingeniería en Telecomunicación. Curso 2013-2014.

Además, la tesis doctoral de **Pablo García Sánchez**, con título *Service Oriented Architecture For Adaptive Evolutionary Algorithms: Implementation And Applications*, cuyos resultados se han utilizado ampliamente en el proyecto. La memoria de la tesis se encuentra disponible en <http://evorq.ugr.es/actas/tesis.pdf>

Y por último, este Trabajo Fin de Máster sobre optimización y difusión en la nube en la predicción del tráfico utilizando tecnologías inalámbricas y métodos de predicción de series temporales.

*Estudia el pasado
siquieres pronosticar el futuro.
— Confucio, 551AC-478AC*

AGRADECIMIENTOS

Resulta cuando menos curioso, que habiendo sido capaz de escribir más de un centenar de páginas para definir el proyecto, me cueste tanto escribir una sola página para los agradecimientos. Antes de nada, hago un `#include(all)` por si me olvido de citar a alguien que quede recogido en dicha sentencia.

A mi familia: mi madre, mis hermanos José Andrés y David y a las familias que a su vez estos van creando Mari Jose y María del Mar¹: por aguantarme aunque la mitad de lo que les cuente le suene a chino y a veces piensen que estoy un poco loco.

A mis compañeros de investigación: a mi jefe² Pedro por darme la oportunidad de trabajar en un proyecto como este y ser de los mejores jefes que he tenido y de las mejores personas que he conocido; A JJ que es el que me tendió una mano para llegara este mundo de la investigación; a Antonio, por la paciencia y dedicación que pone en orientarme en el mundo de la investigación (aunque a veces no llegue a las fechas de los congresos); a Pablo, por ser el reflejo de lo que quiero ver en mi espejo en unos años; a Maribel, porque es una delicia trabajar e ir a las reuniones con ella y porque aprendí (por fin) R con su manual; a Javi compañero de proyecto, porque al ser los dos tan diferentes no dejábamos punto de vista sin contemplar; a Paloma compañera de andanzas en el máster; al resto del grupo de investigación (Gustavo, José Luis, Carlos, Carpio, Pedro G. , Victor, Juanlu, ...) porque entre todos ellos cada día aprendo algo. E incluyo a los también compañeros de la oficina de software libre, que se lo curran.

A mis compañeros tanto en en el despacho: Curro, Oresti, Nuria, Paloma, Antonio, Javi, Pablo por generar un ambiente tan bueno de trabajo; como en todo el CITIC aunque muchos se hayan ido ya a un lugar mejor: Quique, Leo, Raquel, Sara, Nolo, Cristina, Gonzalo,... Soy muy malo para los nombres, así que si me olvido de alguien que no sé preocupe, que está incluido arriba. Bueno, y no sólo a los compañeros sino a todo el personal del CITIC:

¹ ¡No has nacido y ya tienes una cita en un texto científico!

² :)

Paco I., Paco B.H, Bea,... que siempre me ha echado una mano cuando he necesitado cualquier gestión.

A mis amigos, tanto los que están cerca como los que están lejos: Ana G., Ana P. Alberto, Jero, Pelayo, Pili, Rafa, Sergio, ... Joe, que malo soy para poner nombres. Buenos, ellos saben quienes son.

A mis compañeros del máster: Javi, Paloma, Miguel, Alfonso, Pedro, Pablo, Juan, José, Victor, Juan, ... porque yo pensaba que alguno no terminaba el máster vivo. Lo digo por lo de la nieve.

A los compañeros de Ciudad2020, que se han encargado de la parte Hardware del proyecto: Manuel Rodriguez, Manuel Ara-gón, Alfredo Romeo; y a los compañeros de la OPA: Manuel Fernández y Manuel Gonzalez.

A los profesores del máster y compañeros del departamento.

A mis compañeros de piso, que no sabían si yo vivía allí o era un espectro con el que se cruzaban ocasionalmente en el pasillo o la cocina.

Y bueno, un rinconcito friki. A Mercadona, Domino's, Telepizza, Burger King, MacDonalds,...³ por ser la base de mi mala alimentación. A los dueños de la Posada, porque cada vez que vamos nos ponen tapas más grandes y ricas. A Marvel, que queda vez hace mejores películas. A MLP:FIM, gracias a la cual estoy mejorando mi inglés. A Blizzard, porque matar hordas de demonios relaja bastante. A Valve, que ya pueden sacar el Half-Life 3 que ahora espero andar mejor de tiempo.

Y en definitiva a todo el mundo, menos a las señoras mayores que se cuelan en la cola del Mercadona.

³ Aunque también han caído bastantes ensaladas en el despacho, que conste.

LISTA DE CONTENIDOS

i	INTRODUCCIÓN	1
1	ANTECEDENTES	3
1.1	Tecnologías Actuales	5
1.2	Colaboración con la empresa Ciudad 2020	7
2	OBJETIVOS	9
2.1	Elementos en desarrollo	10
2.2	Elementos hardware	11
2.3	Servicios Web y herramientas de información	11
2.4	Objetivos del Proyecto Fin de Máster	12
3	METODOLOGÍA	13
3.1	Po1 - Coordinación, administración y gestión	14
3.2	Po2 - Componentes hardware	14
3.3	Po3 - Sistema de alimentación	15
3.4	Po4 - Infraestructura de servicios web e interoperabilidad	15
3.5	Po5 - Diseño de la inteligencia del negocio	16
3.6	Po6 - Integración del sistema y pruebas	16
3.7	Po7 - Plataforma web	17
3.8	Po8 - Visualización de datos	17
3.9	Po9 - Despliegue y mantenimiento del sistema de información	18
3.10	P10 - Difusión de resultados	18
ii	OPTIMIZACIÓN DEL PROCESADO DE DATOS	21
4	SERVIDOR LOCAL	23
4.1	Hardware	24
4.1.1	RAID	25
4.2	Sistema operativo	26
4.2.1	Configuración adicional del Sistema Operativo: Sistema no-atime	26
4.2.2	Configuración adicional del sistema operativo: Fichero temporales y logs en memoria RAM (Descartado)	27
4.3	Base de datos local: MySQL	28
4.3.1	Deshabilitado de la Query-Caché o caché de consultas	28
4.4	Configuración de APACHE	29
5	SIMA: ACTUALIZADOR LOCAL	31
5.1	Introducción	32

5.2	Descarga de Datos	32
5.2.1	Descarga de Nodos	33
5.2.2	Descarga de Dispositivos y Pasos	34
5.2.3	Paralelizado	36
5.3	Procesado de los Datos	36
5.3.1	Diferentes técnicas de procesado: SPLIT	37
5.3.2	Diferentes técnicas de procesado: JSON	38
5.3.3	Comparativa de diferentes técnicas de procesado	39
5.4	Insercción de los Datos	39
5.4.1	Procesamiento de peticiones en lote	41
5.4.2	Paralelización	42
5.5	Cliente Actualizador DB Local	46
5.5.1	Estructura del módulo	47
5.5.2	Tipos de sincronización	49
5.5.3	Gestión de los nodos	50
5.5.4	Gestión de la ventana de tiempos	50
6	SIMA: GESTIÓN DE LA CONFIGURACIÓN	59
6.1	Uso de ficheros properties para la configuración	60
6.2	Sistema de doble fichero de configuración	62
6.2.1	Ficheros properties encriptados	63
6.3	Uso de módulo de configuración	64
6.4	Parámetros de configuración del sistema	65
6.4.1	Modo Depuración	65
6.4.2	Variables de conexión a la base de datos Local	65
6.4.3	Variables de optimización de las consultas a la base de datos Local	65
6.4.4	Variables de optimización para la concurrencia de las peticiones a la base de datos Local	66
6.4.5	Directorio de configuración y almacén de credenciales	66
6.4.6	Variables de configuración de Fusion Tables	66
6.4.7	Credenciales para el acceso a la información de los nodos	68
6.4.8	Parámetros de configuración del cliente de twitter	68
6.4.9	Constantes geográficas	69
7	SIMA: MECANISMOS DE DEPURACIÓN	71
7.1	Entorno de depuración: Empleo de objetos Logger	72
7.1.1	Formato personalizado de fichero de depuración.	74

7.1.2	Ejemplo de uso	75
7.2	Mediciones de tiempo	76
7.2.1	Ejemplo de uso	77
7.3	Formatos de fecha y números flotantes	77
7.3.1	Ejemplo de uso	78
iii	ALMACENAMIENTO Y OPTIMIZACIÓN DEL ACCESO A DATOS	79
8	BASE DE DATOS LOCAL: MYSQL	81
8.1	Esquema de base de datos	82
8.1.1	Nodo	82
8.1.2	Dispositivos	83
8.1.3	Pasos	83
8.2	Optimizaciones en la base de datos	84
8.2.1	Optimizado de consultas con ventanas temporales	85
8.2.2	Optimización de consultas de recurrencias	88
8.2.3	Particionado de datos	90
8.2.4	Reducción del tamaño de las bases de datos	93
8.3	Procedimientos de procesamiento de datos	95
8.3.1	Procedimiento: Agrupación Pasos Por intervalos	95
8.3.2	Procedimiento: Cálculo de trazas para pasos entre nodos	98
9	BASE DE DATOS EN LA NUBE: GOOGLE FUSION TABLES	101
9.1	Ventajas de uso de Google Fusion Tables	102
9.1.1	Defición	102
9.1.2	Beneficios de uso	103
9.2	Google Fusion Tables	105
9.2.1	Arquitectura y almacenamiento de datos	105
9.2.2	Procesado de consultas	109
9.2.3	Transacciones	109
9.2.4	Visualización de los datos	109
9.2.5	Características geolocalizadas	110
9.2.6	Integración con aplicaciones de terceros	110
9.3	Uso de Fusion Tables en el Proyecto	110
iv	PREDICCIÓN	111
10	PREDICCIÓN OFFLINE	113
10.1	Estudio preliminar series temporales	114
10.1.1	ARIMA	115
10.1.2	L-Co-R	116
10.1.3	Weka Forecasting	117
10.2	Predicción por días	117

10.3	Predicción por horas	121
11	PREDICCIÓN ONLINE	125
11.1	Software empleado	126
11.1.1	R	126
11.1.2	Weka	128
11.2	SiMa: Módulo de Predicción	130
11.2.1	Carga de datos	130
11.2.2	Construcción del modelo de predicción	131
11.2.3	Controlador del Módulo	134
V	PUBLICACIÓN Y DIFUSIÓN EN LA NUBE	137
12	SIMA: ACTUALIZADOR EN LA NUBE CON GOOGLE FUSION TABLES	139
12.1	Autorización	140
12.2	Optimización de procesado: Sistema de colas	141
12.3	Métodos desarrollados para trabajar con métodos SQL	144
12.3.1	Métodos de selección: SELECT	145
12.3.2	Métodos de inserción: INSERT	145
12.3.3	Métodos de actualización: UPDATE	151
12.3.4	Métodos de borrado: DELETE	151
12.4	Sistemas de Actualización	152
12.4.1	Actualización de Nodos	152
12.4.2	Actualización de Pasos por Horas	153
12.4.3	Actualización de Trazas por Horas	155
12.5	Cliente Actualizador FT	157
13	SIRVIENDO LOS DATOS: GOOGLE FUSION TABLES API REST	159
13.1	Referencia rápida de la API REST	160
13.1.1	Consultas sobre los datos: SELECT	160
13.1.2	Insercción sobre los datos: INSERT	162
13.1.3	Actualizaciones sobre los datos: UPDATE	163
13.1.4	Borrado de datos: DELETE	164
13.2	Ejemplo de uso	164
13.2.1	Impresión en tabla HTML	165
13.2.2	Mapa de Google Maps	166
14	DESARROLLO WEB Y MÓVIL	169
14.1	Servicios Web	170
14.1.1	Tablas	170
14.1.2	Gráficos	171
14.1.3	Mapas	172
14.2	Plataformas Web	177
14.2.1	Página principal: Wordpress	177
14.2.2	Visor de eventos online	179
14.2.3	Panel de control	180

14.2.4	Panel de Administración de Google Fusion Tables	183
14.3	Aplicación Móvil	184
14.3.1	Optimizaciones	184
15	REDES SOCIALES, CIENCIA ABIERTA Y DIFUSIÓN	187
15.1	Redes Sociales	188
15.1.1	Twitter	188
15.1.2	Facebook	190
15.1.3	Youtube y Google Plus	191
15.2	Ciencia Abierta	192
15.2.1	Github	192
15.3	Difusión en prensa del proyecto	192
vi	RESULTADOS Y CONCLUSIONES	195
16	CONCLUSIONES	197
17	FUTURAS LÍNEAS DE INVESTIGACIÓN	201
vii	ANEXOS	203
A	BLUETOOTH	205
A.1	Service Class: Clase de servicio	206
A.2	Clases de dispositivo	206
A.2.1	Major Device Class	206
A.2.2	Minor Device Classes	207
A.2.3	Toy Major Class	209
	BIBLIOGRAFÍA	211

LISTA DE FIGURAS

Figura 1.1	Clasificación de los sistemas de información para transporte en función de la intrusividad de la tecnología	6
Figura 5.1	Tareas del sincronizador local de datos	32
Figura 5.2	Dependencia entre los datos	32
Figura 5.3	Código: Descarga de nodos	33
Figura 5.4	Código: Descarga de dispositivos y pasos: constructor	34
Figura 5.5	Código: Descarga de dispositivos y pasos: creador del recurso	35
Figura 5.6	Código: Descarga de dispositivos y pasos: realizar descarga	35
Figura 5.7	Código: Procesado: método SPLIT	37
Figura 5.8	Código: Procesado: método SPLIT	38
Figura 5.9	Comparativa en tiempos de los distintos métodos de procesado	39
Figura 5.10	Código: Versión preliminar método de insertado	40
Figura 5.11	Código: Inserción por lotes	41
Figura 5.12	Comparativa en tiempos de los distintos métodos de procesado	42
Figura 5.13	Código: Clase encargada del procesamiento paralelo de la inserción en la base de datos	43
Figura 5.14	Código: Métodos de la clase de inserción paralelo	44
Figura 5.15	Código: Método de inserción paralelo	45
Figura 5.16	Código: Clase encargada de la gestión de las hebras de inserción	46
Figura 5.17	Código: Estructura del módulo	48
Figura 5.18	Ejecución del módulo	49
Figura 5.19	Código: Función actualizarNodos	50
Figura 5.20	Código: Función actualizaDesdeFecha I	52
Figura 5.21	Código: Función checkChildrens	54
Figura 5.22	Código: Función whaitForIt	54
Figura 5.23	Código: Función actualizaDesdeFecha II	56
Figura 6.1	Código: Inicialización del entorno: Configuración	60
Figura 6.2	Código: Clase encargada de la gestión de la configuración I	61

Figura 6.3	Configuración: Jerarquía de búsqueda	62
Figura 6.4	Código: Clase encargada de la gestión de la configuración II	63
Figura 6.5	Código: Clase encargada de la gestión de la configuración III	64
Figura 7.1	Código: Inicialización del entorno: Depuración	72
Figura 7.2	Código: Inicialización del entorno: Configuración del sistema de Depuración	73
Figura 7.3	Código: Formato personalizado de depuración	75
Figura 7.4	Código: Uso del sistema de depuración	75
Figura 7.5	Código: Sistema de depuración: mediciones de tiempo	76
Figura 7.6	Código: Sistema de depuración: mediciones de tiempo (ejemplo de uso)	77
Figura 7.7	Código: Sistema de depuración: formateo de fechas y decimales	78
Figura 7.8	Código: Sistema de depuración: formateo de fechas y decimales	78
Figura 8.1	Esquema de la base de datos local: Nodos	82
Figura 8.2	Esquema de la base de datos local: Dispositivos	83
Figura 8.3	Esquema de la base de datos local: Dispositivos	84
Figura 8.4	Funcionamiento de un índice Hash en la localización de tuplas	85
Figura 8.5	Funcionamiento de un índice basado en BTREE en la localización de tuplas	86
Figura 8.6	Consulta a ejecutar para la experimentación con índices	87
Figura 8.7	Consulta a ejecutar para la experimentación con índices II	87
Figura 8.8	Consulta a ejecutar para la experimentación con índices III	89
Figura 8.9	Almacenamiento de una tabla sin realizar particionado	91
Figura 8.10	Funcionamiento de un tabla particionada en función del idNodo.	92
Figura 8.11	Procedimiento agrupamiento pasos por intervalos I	96
Figura 8.12	Procedimiento agrupamiento pasos por intervalos II	97
Figura 8.13	Procedimiento agrupamiento pasos por intervalos III	97

- Figura 8.14 Procedimiento calculo de trazas **99**
Figura 9.1 Arquitectura de almacenamiento de Google Fusion Tables **106**
Figura 10.1 Tareas del método de predicción ARIMA **115**
Figura 10.2 Serie temporal de nodos para 60 días **118**
Figura 10.3 Total vehículos por día de la semana para la serie de 60 días **119**
Figura 10.4 Serie temporal de nodos por horas **121**
Figura 10.5 Valores predichos para 24 horas de las series A,B,C **124**
Figura 11.1 Código: Código para la predicción en R: Script **127**
Figura 11.2 Código: Código para la predicción en R: Java **127**
Figura 11.3 Gráfica de salida de predicción en R **128**
Figura 11.4 Código: Código para la predicción en Weka **129**
Figura 11.5 Código: Carga de datos en Weka desde memoria **131**
Figura 11.6 Código: Predicción mediante Weka **133**
Figura 11.7 Código: Actualizador de Predicción I **134**
Figura 11.8 Código: Actualizador de Predicción II **135**
Figura 12.1 Código: Código para la gestión de credenciales de conexión a Google Fusion Tables **141**
Figura 12.2 Código: Código para la gestión de colas para Fusion Tables I **142**
Figura 12.3 Código: Código para la gestión de colas para Fusion Tables II **143**
Figura 12.4 Código: Métodos para Fusion Table: SQL **144**
Figura 12.5 Código: Métodos para Fusion Table: SELECT **145**
Figura 12.6 Código: Métodos para Fusion Table: INSERT Sobrecarga **146**
Figura 12.7 Código: Métodos para Fusion Table: INSERT **148**
Figura 12.8 Código: Métodos para Fusion Table: INSERT II **150**
Figura 12.9 Código: Métodos para Fusion Table: UPDATE **151**
Figura 12.10 Código: Métodos para Fusion Table: DELETE **152**
Figura 12.11 Código: Fusion Tables: Subsistema de subida de nodos **153**
Figura 12.12 Código: Fusion Tables: Subsistema de subida de Pasos por hora I **154**

- Figura 12.13 Código: Fusion Tables: Subsistema de subida de Pasos por hora II 155
- Figura 12.14 Código: Fusion Tables: Subsistema de subida de Trazas por hora 156
- Figura 12.15 Código: Fusion Tables: Cliente Actualizador de FT 157
- Figura 12.16 Código: Fusion Tables: Cliente Actualizador de FT: Temporizador 158
- Figura 13.1 Google Fusion Tables API: Select 160
- Figura 13.2 Google Fusion Tables API: INSERT 162
- Figura 13.3 Google Fusion Tables API: UPDATE 163
- Figura 13.4 Google Fusion Tables API: DELETE 164
- Figura 13.5 Código: Ejemplo de uso de la API REST de Google Fusion Tables 165
- Figura 13.6 Código: Ejemplo de uso de Google Fusion Tables en Google Maps. 166
- Figura 13.7 Código: Ejemplo de uso de la API REST de Google Fusion Tables en Google Maps. 167
- Figura 14.1 Representación en modo de tabla de los datos 171
- Figura 14.2 Gráfico de serie temporal 172
- Figura 14.3 Animación de la gráfica interativa 173
- Figura 14.4 Mapa principal de visualización de los datos 174
- Figura 14.5 Mapa principal de visualización de los datos: Zonas de control 174
- Figura 14.6 Mapa principal de visualización de los datos: Selector de fecha 175
- Figura 14.7 Mapa principal de visualización de los datos: Información del nodo 175
- Figura 14.8 Mapa principal de visualización de los datos: Selector de hora 176
- Figura 14.9 Mapa principal de visualización de los datos: Opciones 176
- Figura 14.10 Web principal: Vista de escritorio 178
- Figura 14.11 Web principal: Vista tablet y móvil 178
- Figura 14.12 Visor de eventos 179
- Figura 14.13 Panel de control 181
- Figura 14.14 Panel de control: Distintos temas 182
- Figura 14.15 Panel de Administración Google Fusion Tables 183
- Figura 14.16 Aplicación móvil 185
- Figura 15.1 Tweets publicados de forma automática por el sistema 189

Figura 15.2	Código: Publicación en Twitter: Servicio privado PHP	189
Figura 15.3	Código: Publicación en Twitter: Java	190
Figura 15.4	Código: Publicación en Twitter: Java	190
Figura 15.5	Página de Facebook del proyecto SIPEsCA	191
Figura 15.6	Página de Youtube del proyecto SIPEsCA	192

LISTA DE TABLAS

Tabla 4.1	Características Hardware del Servidor de Cómputo	24
Tabla 4.2	Tiempos de lectura en disco en un sistema por defecto y en un sistema no-atime	27
Tabla 8.1	Tiempos en la comparativa entre índice por defecto e índice basado en B-TREE	87
Tabla 8.2	Tiempos en la comparativa entre índice por defecto e índice basado en B-TREE II	88
Tabla 8.3	Tamaños asociados a la tabla paso sin optimizaciones	93
Tabla 8.4	Tamaños asociados a la tabla paso con las optimizaciones	93
Tabla 8.5	Tamaños asociados a la tabla paso con las optimizaciones y reducido el tamaño por tupla	94
Tabla 8.6	Tamaños asociados a la tabla paso con las optimizaciones, reducido el tamaño por tupla e implementado en nuevo índice	95
Tabla 10.1	Métricas de error cometido por método en cada nodo	120
Tabla 10.2	Métricas de error para las series temporales A, B y C. Se presenta en negrita el mejor método para cada métrica de error. El mejor valor es siempre el más bajo, salvo en el caso de la métrica DA, en cuyo caso es el mayor.	123
Tabla A.1	Anexo: Bluetooth - Clases de servicio	206
Tabla A.2	Anexo: Bluetooth - Major Device Class	207
Tabla A.3	Anexo: Bluetooth - Phone Major Class	207
Tabla A.4	Anexo: Bluetooth - Audio/Video Major Class	208
Tabla A.5	Anexo: Bluetooth - Peripheral Major Class	208
Tabla A.6	Anexo: Bluetooth - Imaging Major Class	209

Tabla A.7	Anexo: Bluetooth - Wereable Major Class	209
Tabla A.8	Anexo: Bluetooth - Toy Major Class	209

ACRÓNIMOS

API	Application Programming Interface
EA	Evolutionary Algorithm
JSON	JavaScript Object Notation
OAUTH	Open Authorization
PHP	Hypertext Pre-processor
RAID	Redundant Array of Independent Disks
REST	Representational State Transfer
WEKA	Waikato Environment for Knowledge Analysis
XML	eXtensible Markup Language
MAE	Mean Absolute Error
RMSE	Root mean squared error
MAPE	Mean absolute percentage error
RRSE	Root relative squared error
RAE	Relative absolute error
DA	Direction accuracy
MSE	Mean squared error

TÉRMINOS EMPLEADOS EN ESTE DOCUMENTO

nodo	Sistema remoto de bajo coste de adquisición de paso de vehículos. Dispone de una posición geográfica y un tipo de sensor de adquisición
dispositivo	Un dispositivo es un vehículo, persona o ente con un identificador único que es almacenado en el sistema

paso	Captación de un nodo de un dispositivo en un instante de tiempo concreto
traza	Reocurrencia de captación de un mismo dispositivo en dos nodos distintos en un corto periodo de tiempo. Se considera por tanto que el dispositivo a viajado desde un nodo al otro
SiMa	Nombre bajo el que se engloba todo el entramado software desarrollado para el proyecto SipeSCA para el procesamiento local, predicción y publicación en la nube

Parte I

INTRODUCCIÓN

1

ANTECEDENTES

ÍNDICE DE CAPÍTULO

1.1	Tecnologías Actuales	5
1.2	Colaboración con la empresa Ciudad 2020	7

La mayoría de los sistemas empleados para monitorizar el flujo de tráfico en las carreteras resultan excesivamente caros para una implantación en masa, y además solo recogen información sobre la cantidad de vehículos que han circulado. Además, esas tecnologías son incapaces de identificar el mismo vehículo en diferentes sitios u ocurrencias sobre el mismo dispositivo capturador, limitándose tan solo a contar y obtener algún tipo de información vaga sobre el tipo de vehículo [6, 32].

Contar con un sistema de información sobre el estado del tráfico y la predicción del uso de la red viaria por parte de los vehículos se antoja clave en el contexto actual. Con una población cada vez más informada y con dispositivos de comunicación ubicuos que poseen y usan habitualmente prácticamente el 90% de la población, obtener información sobre cómo se encuentra el tráfico en cualquier momento en cualquiera de los casi 20.000 kilómetros con los que cuenta la red viaria nacional, significaría poder gestionar de manera óptima una red de comunicaciones vital para un porcentaje elevado de usuarios.

Tal sistema ayudaría a las políticas urbanísticas y de ordenación territorial coordinadas con las de movilidad al reducir la necesidad de desplazamiento al poder planificar los ciudadanos sus desplazamientos (incluso en transporte público).

La aplicación de esta propuesta en el transporte supondría disponer de un sistema de información sobre el estado del tráfico y de predicción del uso de la red viaria por parte de los vehículos. También ayudaría a gestionar de manera óptima una red de comunicaciones vital para un porcentaje elevado de usuarios.

La primera consecuencia directa sería un aprovechamiento más eficiente del uso de los vehículos en los desplazamientos, por lo que se podría esperar un menor número de trayectos en vehículo privado, o al menos más optimizados.

Estos objetivos son los que nos hemos planteado en el presente desarrollo, y pasan por contar con un sistema de información y de predicción de tráfico para que puedan ser utilizados con el fin de reducir los desplazamientos individuales, así como incrementar los desplazamientos en transporte público.

Nuestro objetivo final es disponer de información acerca de los flujos de tráfico que se producen entre ciudades, lo que permitirá poder gestionar de manera óptima las decisiones de desplazamiento por parte de los ciudadanos.

La idea es obtener un sistema de ayuda a la toma de decisiones, capaz de aplicar conocimiento en aplicaciones comerciales relacionadas con la movilidad.

Para ello, el procesamiento de los datos adquiridos con estos sistemas requiere de algoritmos complejos de minería de datos, computación evolutiva y redes neuronales, aprendizaje automático y de métodos estadísticos, que se irán desarrollando e integrando como servicios web en el sistema.

Por tanto, encontramos que se tienen varias necesidades desde el punto de vista de la gestión del transporte:

- Se necesitará un dispositivo autónomo y versátil de recogida de datos y monitorización.
- Además es necesario recopilar los datos del tráfico en tiempo real.
- Una vez se tienen los datos, hay que procesarlos de manera correcta para poder ofrecer la información específica necesaria.
- También se necesita un sistema que teniendo en cuenta la evolución de los actuales sistemas de información, permita poder compartir esos datos con aquellos que toman decisiones sobre movilidad, no sólo desde el punto de vista institucional sino también desde el punto de vista personal.
- Por último, y máxime en el contexto actual, se necesita un sistema de información que sea de bajo coste, que permita su amortización de la manera más rápida posible y que posibilite que sea capitalizado por parte de todo tipo de agentes en la ciudad.

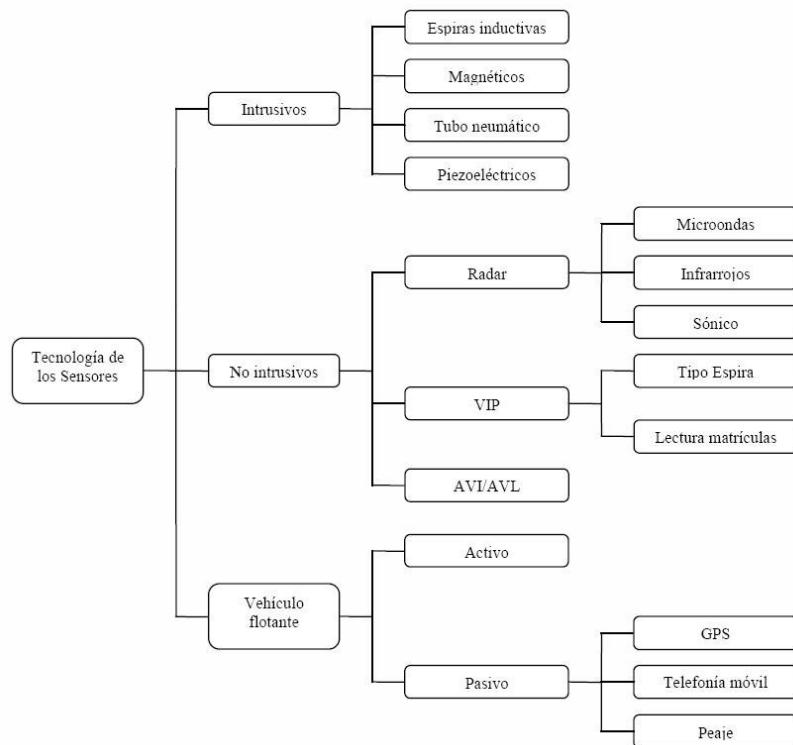
1.1 TECNOLOGÍAS ACTUALES

Los sistemas de información aplicados a la recopilación de datos y generación de información sobre el estado de las carreteras se clasifican por la inmediatez en la obtención de los datos (la toma de datos se define como directa cuando la fuente obtenga el dato estudiado sin utilizar algoritmos adicionales), por la exhaustividad en la recopilación (la toma de datos será casi exhaustiva, cuando su representatividad sea casi total, es decir cuando el número de medidas tomadas coincida con el número de usuarios de ese tramo de la red) y por la intrusividad que tengan estos. La Figura 1 muestra una clasificación de los sistemas de información en función de la intrusividad de la tecnología.

Principalmente, los sistemas utilizados en la actualidad son:

- tubos neumáticos o aforadores (sensores de ejes que detectan el paso del vehículo),

Figura 1.1
Clasificación de los sistemas de información para transporte en función de la intrusividad de la tecnología



- detectores de lazo o espiras (detectan el paso del vehículo por variación de la masa magnética sobre el lazo),
- vehículos flotantes (dotados de sensores que recoge información mientras circula por una ruta predefinida), y
- sistemas de identificación automática de vehículos (en los telepeajes).

La principal desventaja de estos sistemas es que no tienen capacidad para identificar e individualizar los vehículos que detectan, por lo que no es posible la elaboración de matrices origen/destino dinámicas con infraestructuras basadas en ellos.

Sólo es posible saber que han pasado un número de vehículos y la tipología de los mismos, pero no permite conocer los flujos de movimiento tal y como se producen.

Además, su coste elevado hace muy poco rentable cubrir la red de carreteras secundarias con ellos, por lo que se suelen ubicar en vías principales y en salidas de grandes núcleos de población.

1.2 COLABORACIÓN CON LA EMPRESA CIUDAD 2020

Ciudad 2020 es una empresa TIC de Córdoba cuya misión es conectar la realidad del entorno, para poder medirla, analizar y tomar decisiones en función de la información extraída. Su principal objetivo es organizar la información del entorno físico para ayudar en la toma de decisiones a todo tipo de organizaciones en función del análisis del flujo y comportamiento de las personas. Trabaja principalmente en la Internet de las cosas, sociometría y reality mining.

La tecnología desarrollada por Ciudad 2020 y los servicios que ofrece se basan en la implantación de una red de dispositivos Intelify repartidos por la ciudad para obtener los datos de movilidad.

Mediante un despliegue de dispositivos autónomos por la ciudad, ofrecen información de importancia entre otros para los sectores turístico y comercial, así como para la movilidad en la ciudad. Concretamente, en se ofrece un servicio de monitorización del paso de personas por las calles céntricas de la ciudad de Córdoba.

2

OBJETIVOS

ÍNDICE DE CAPÍTULO

2.1	Elementos en desarollo	10
2.2	Elementos hardware	11
2.3	Servicios Web y herramientas de información	11
2.4	Objetivos del Proyecto Fin de Máster	12

El objetivo principal del proyecto SIPEsCA en el que se enmarca este proyecto Fin de Máster es conseguir un sistema de información autónomo (desde un punto de vista energético), que sea de bajo coste, de rápida implantación y de alta fiabilidad, tal que informe sobre las condiciones del tráfico en tiempo real, no sólo para las instituciones y organismos encargados de la regulación y control del tráfico, sino también a usuarios particulares (a través de alertas móviles, mediante web, etc.), y que gracias a la recopilación y procesado de datos permita predecir el comportamiento futuro.

El proyecto Fin de Máster que se defiende en este documento se encarga de la elaboración del sistema encargado del procesamiento de los datos y la publicación de los resultados obtenidos en la Nube. Abarca también los aspectos relacionados con la optimización del acceso a los datos y el desarrollo de los servicios y plataformas web que permiten a los usuarios el acceso a los datos.

Se recogen a continuación los objetivos globales del proyecto, marcando con (*) aquellos que se encuentran directamente relacionados y recogidos en este trabajo fin de máster.

2.1 ELEMENTOS EN DESARROLLO

Al comienzo del proyecto SIPEsCA existen diferentes elementos del sistema de información propuesto que ya están en desarrollo:

- Sistema de recopilación de datos de los dispositivos: se han estudiado tres posibilidades: escaneo de dispositivos Bluetooth, dispositivos WiFi e implantación de tags RFID en vehículos. Así, el dispositivo autónomo montará estos dispositivos para la captura de frecuencias a altas velocidades (se utilizarán y configurarán tanto en hardware como en software). Adicionalmente el sistema usa una conexión 3G para enviar los datos obtenidos de los demás sensores al servidor de almacenamiento.
- Sistema de alimentación autónomo: Se ha elegido y personalizado una caja para alojar la tecnología que resista todo tipo de condiciones metereológicas y de posibles actos vandálicos y que deberá ser ubicada en los márgenes de las carreteras a monitorizar. Junto con la caja, se está desarrollando un sistema de alimentación autónomo haciendo uso de energías renovables y autónomas como luz solar, pilas, etc.

2.2 ELEMENTOS HARDWARE

Los objetivos anteriormente fijados se están desarrollando, desde el punto de vista hardware, mediante:

- El desarrollo e implantación de 16 dispositivos autónomos (con capacidad para detectar señales BT y wifi) para la recopilación de la información sobre la movilidad, encargados de enviar la información a los servidores para el posterior procesamiento de la información.
- La configuración del sistema montado en un vehículo de transporte para hacer pruebas (equipado con un dispositivo móvil con Bluetooth y Wifi activado, además de un chip de medio/largo alcance para la identificación).
- A este conjunto de dispositivos se suman los 6 que tiene el grupo de investigación instalados en Granada.

2.3 SERVICIOS WEB Y HERRAMIENTAS DE INFORMACIÓN

Desde el punto de vista de servicios web y herramientas de información, están en desarrollo:

- Sistema de procesado de datos para almacenar correctamente todos los datos generados y servirlos a través de un servicio web. (*)
- Servicio de información para facilitar toda la información a los usuarios interesados en conocer el estado de las carreteras. (*)
- Configuración de los elementos de software necesarios para la predicción mediante mecanismos de inteligencia artificial de los comportamientos futuros de los usuarios del transporte. (*)
- Website con un panel de información que permitirá conocer los principales datos de movilidad en tiempo real a partir de la información recopilada. (*)
- Servicio de alertas automatizado y en tiempo real cuando se cumplan una serie de condiciones dadas. (*)
- Servicio web para la consulta y extracción de datos por parte de sistemas de información de terceros en tiempo real del sistema (*)
- Servicio de predicción de los flujos de movimiento. (*)

2.4 OBJETIVOS DEL PROYECTO FIN DE MÁSTER

Si bien anteriormente se han presentado todos los objetivos del proyecto SIPEsCA en este proyecto se abarcan únicamente los objetivos relacionados con los servicios Web y Herramientas de información.

Si bien el hardware ha sido una pieza de desarrollo vital en este proyecto, al haber sido una pieza no desarrollada por el alumno, no se le realizará mayor mención que en este apartado ¹.

Es por tanto en el sistema y métodos presentados en este Proyecto Fin de Máster totalmente independiente del hardware que produce dichos datos. Se presenta así por tanto el sistema SiMa desarrollado para la adquisición, procesamiento y publicación de los datos del proyecto SIPEsCA.

Dicho sistema se encuentra realizado en JAVA para la capa de lógica de aplicación, debido a que es multiplataforma, a la gran cantidad de librerías existentes y a su capacidad de compaginar distintos tipos de metodología como la programación orientada a objetos, eventos y sistemas paralelos.

Si bien algunos autores consideran a JAVA “lento” debido al uso de JVM y su proceso de compilación JIT² es necesario reconocer que este método de compilación no es el actualmente utilizado por las máquinas, que realizan la primera compilación de los bytecodes a código máquina. De esta forma, si bien la “puesta en marcha” del sistema no sea la solución más eficiente, una vez iniciado el sistema (cosa que sólo se ha realizado un par de veces) el sistema funciona con una eficiencia similar a la de lenguajes de más bajo nivel.

No por nada, JAVA es de los sistemas más empleados en el mundo empresarial para sistemas donde se requiera una alta tolerancia a fallos y entornos 24/7.

Para la capa de representación de datos, optamos por la tecnología Web debido a ser el estándar más común

¹ Queremos en documento agradecer la labor de la empresa *Ciudad2020* la excelente labor realizada con el hardware del proyecto SIPEsCA.

² Just in time

3

METODOLOGÍA

ÍNDICE DE CAPÍTULO

- 3.1 Po1 - Coordinación, administración y gestión **14**
 - 3.2 Po2 - Componentes hardware **14**
 - 3.3 Po3 - Sistema de alimentación **15**
 - 3.4 Po4 - Infraestructura de servicios web e interoperabilidad **15**
 - 3.5 Po5 - Diseño de la inteligencia del negocio **16**
 - 3.6 Po6 - Integración del sistema y pruebas **16**
 - 3.7 Po7 - Plataforma web **17**
 - 3.8 Po8 - Visualización de datos **17**
 - 3.9 Po9 - Despliegue y mantenimiento del sistema de información **18**
 - 3.10 Po10 - Difusión de resultados **18**
-

Al estar este Proyecto Fin de Máster enmarcado en un proyecto de una magnitud mucho mayor, han sido muchas las personas (tanto técnicas como investigadoras) involucradas en el proyecto SIPEsCA.

Se recoge en este capítulo la carga de tareas en las que se haya dividido el proyecto, haciendo especial distinción de aquellas tareas relacionadas con el desempeño de este Proyecto Fin de Máster.

A continuación se describen los avances de los trabajos, indicando las personas involucradas en dichos trabajos.

3.1 P01 - COORDINACIÓN, ADMINISTRACIÓN Y GESTIÓN

- Continúan las tareas de coordinación y gestión del proyecto, así como las de diseminación y publicación de resultados.
- Se ha renovado el contrato de la primera persona contratada, que está trabajando en la administración del servidor.
- También se han subcontratado diversos trabajos a la empresa Ciudad2020.
- Se han gestionado los permisos de colocación de los dispositivos para su colocación en el área metropolitana de Sevilla.
- Se ha ofertado un segundo contrato de personal hasta el final del proyecto.
- Falta por subcontratar los últimos trabajos a la empresa Ciudad2020.

Personas involucradas en el desarrollo del P01

Todos los investigadores del equipo, con la colaboración de los técnicos de Ciudad 2020 y el personal de la AOPA.

3.2 P02 - COMPONENTES HARDWARE

- Se han estudiado diferentes tecnologías y se han probado varios prototipos para implementar los dispositivos de monitorización.
- Se han configurado varios puntos wifi.
- Se han actualizado los dispositivos a un modelo basado en la plataforma Raspberry-Pi.

- Se han realizado diversos ajustes en las configuraciones para mejorar el funcionamiento.
- Se ha implementado la tecnología basada en tags RFID para su monitorización y la evaluación de esta tecnología.

Personas involucradas en el desarrollo del P02

Los técnicos de Ciudad 2020 con la supervisión de PGF, GRL y PACV.

3.3 PO3 - SISTEMA DE ALIMENTACIÓN

- Se han estudiado las opciones disponibles para alimentar los dispositivos de monitorización.
- Se ha desarrollado una opción basada en alimentación continua y otra basada en paneles solares y batería.
- Se han instalado ambos tipos de dispositivos (alimentaciones) para evaluarlas y extraer conclusiones.

Personas involucradas en el desarrollo del P03

Los técnicos de Ciudad 2020.

3.4 PO4 - INFRAESTRUCTURA DE SERVICIOS WEB E INTEROPERABILIDAD

- Se ha diseñado la base de datos (que está en continua actualización). (*)
- Se diseñó una primera versión de las consultas a la BD. (*)
- Se realizaron optimizaciones tanto a la BD como a las consultas. (*)
- Se realizó la adquisición, instalación y configuración de servidores. (*)
- Se realiza el mantenimiento periódico de los servidores. (*)
- Se han configurado diversas tablas de BD en Google Fusion Tables para servir la información de forma ágil y asegurando la disponibilidad de la información. (*)

- Se ha desarrollado la interfaz de comunicaciones con el servidor de almacenamiento, basada en servicios REST. (*)
- Se han desarrollado y configurado los primeros prototipos de servicios web para servir información a los usuarios. (*)
- Se ha desarrollado una aplicación móvil para facilitar el acceso a los usuarios a toda la información desde cualquier tipo de dispositivo móvil. (*)

Personas involucradas en el desarrollo del P04

Los técnicos de Ciudad 2020, AJFA, MGA, PGS, VMRS y PACV.

3.5 P05 - DISEÑO DE LA INTELIGENCIA DEL NEGOCIO

- Este paquete se ocupa principalmente de la redacción de las especificaciones del sistema.
- Se ha redactado un documento de estado del arte en predicción de series temporales que sirve de base para las publicaciones científicas.
- Se ha recopilado un conjunto de herramientas disponibles gratuitamente para llevar a cabo tareas de data-mining. (*)
- Se han probado varios métodos de predicción y de análisis estadístico para extraer información de los conjuntos de datos. (*)

Personas involucradas en el desarrollo del P05

AMMG, PGF, GRL, JJMG, JJAM, MGA, PGS, VMRS, AJFA y PACV.

3.6 P06 - INTEGRACIÓN DEL SISTEMA Y PRUEBAS

- Se ha desarrollado y configurado una primera versión del servidor (prototipo).
- Se han integrado los algoritmos de procesamiento de datos desarrollados en el mismo.
- El resto de componentes software se han integrado, dando lugar a un prototipo de servidor ya utilizable.

Personas involucradas en el desarrollo del P06

Los técnicos de Ciudad 2020.

3.7 P07 - PLATAFORMA WEB

- Se ha configurado el servidor para gestionar diversos tipos de usuario y servir la información. (*)
- Se llevó a cabo la apertura de la web, tanto para monitorizar los dispositivos de captación de datos, como para servir la información a los usuarios. (*)
- Se han realizado diversas reuniones (virtuales, telefónicas y presenciales) entre las personas involucradas en ciertas tareas, analizando el sistema de información y requisitos iniciales. (*)
- Se ha hecho un diseño del sitio, así como el desarrollo. (*)

Personas involucradas en el desarrollo del P07

los técnicos de Ciudad 2020 junto con PACV.

3.8 P08 - VISUALIZACIÓN DE DATOS

- Las tareas correspondientes a este paquete están en continuo desarrollo. (*)
- Se está usando Google Fussion Tables y otras APIs, y ya se sirve información dinámica e interactivamente. (*)
- Se han realizado diversas reuniones (virtuales, telefónicas y presenciales) entre las personas involucradas en ciertas tareas para analizar las necesidades de visualización y determinar las necesidades de los usuarios.
- Se ha realizado la maquetación de la información para servirla a los usuarios. (*)
- Se han realizado baterías de pruebas con los investigadores, usando diversos tipos de dispositivos, navegadores y sistemas operativos. (*)

Personas involucradas en el desarrollo del P08

Los técnicos de Ciudad 2020 junto con AMMF y PACV.

3.9 P09 - DESPLIEGUE Y MANTENIMIENTO DEL SISTEMA DE INFORMACIÓN

- Se han llevado a cabo los estudios de localización, tanto en Córdoba como en Sevilla.
- Se han obtenido los permisos para desplegar los dispositivos en el área metropolitana de Sevilla, en carreteras gestionadas por la Junta de Andalucía.
- Se han colocado los dispositivos sin necesidad de realizar obra civil.
- Inicialmente se recopilaban datos usando cinco dispositivos. Finalmente se instalaron todos los dispositivos del proyecto y sirven datos continuamente.
- Los sistemas se han desarrollado de forma escalable para que puedan trabajar con un número más alto de nodos en un futuro.
- Se han elegido tecnologías escalables y fiables que aseguren la disponibilidad del sistema.

Personas involucradas en el desarrollo del P09

Los técnicos de Ciudad 2020 junto con el personal de la AOPA y PACV

3.10 P10 - DIFUSIÓN DE RESULTADOS

- Continúan las tareas de presencia en los medios de comunicación.
- Continúan las tareas de publicación científica. Hasta la fecha se han publicado 19 publicaciones en congresos y revistas internacionales. (*)
- Asesorados por un experto, se ha reestructurado el repositorio en GitHub para liberar los datos y los programas de acuerdo a una licencia de software libre. (*)

Personas involucradas en el desarrollo del P10

AJFA, PGS, PACV, JJMG, PGF, JJAM y VMRS.

Parte II

OPTIMIZACIÓN DEL PROCESADO DE DATOS

4

SERVIDOR LOCAL

ÍNDICE DE CAPÍTULO

- 4.1 Hardware **24**
 - 4.1.1 RAID **25**
 - 4.2 Sistema operativo **26**
 - 4.2.1 Configuración adicional del Sistema Operativo:
Sistema no-atime **26**
 - 4.2.2 Configuración adicional del sistema operativo:
Fichero temporales y logs en memoria RAM (Des-
cartado) **27**
 - 4.3 Base de datos local: MySQL **28**
 - 4.3.1 Deshabilitado de la Query-Caché o caché de con-
sultas **28**
 - 4.4 Configuración de APACHE **29**
-

En este capítulo se describe la configuración y prestaciones del servidor de cómputo local relacionado con el proyecto SIPEsCA. Debido a la naturaleza en Tiempo Real del procesado de datos del proyecto y el volumen de datos esperable, consideramos necesario hacer especial mención al servidor de cómputo, ya que tanto el diseño de su configuración como las optimizaciones llevadas a cabo en él han supuesto mejoras significativas en el procesado de los datos.

Además, debido a la naturaleza del proyecto, el procesado de los datos ha de hacerse en Tiempo Real, esto es, es necesario un servidor robusto y tolerable a fallos para adquirir los datos de forma constante.

Esta característica, limita también la experimentación de las configuraciones ya que una vez se ha instalado y configurado el servidor, realizar cualquier cambio u optimización sobre él se vuelve muy costosa. Además, de que en muchas ocasiones impone la correcta métrica experimental de las mejoras conseguidas.

4.1 HARDWARE

Las características principales del Hardware del servidor de cómputo son:

Procesador Intel(R) Core(TM) i5-4430 CPU @ 3.00GHz

Memoria 16 GiB DDR3 1333MHz

Tarjeta Gráfica NVIDIA GT 630

Disco duro 1 Western Digital 500GB 7200RPM

Disco duro 2 Western Digital 500GB 7200RPM

Tabla 4.1
Características Hardware del Servidor de Cómputo

Destacar la adquisición de 16GiB de memoria RAM, inversión que permitió mantener en memoria la mayoría de los datos, con la correspondiente mejora en los tiempos de acceso a los mismos. Además, se posibilita la adquisición de otros 16GiB de memoria al quedar dos sockets libres en la placa base.

Se eligió también una tarjeta gráfica marca NVIDIA, por su posibilidad de utilización de la tecnología CUDA como futura línea de investigación.

4.1.1 RAID

En el servidor de computo debíamos de garantizar la seguridad e integridad de los datos, además de intentar mejorar en lo máximo posible las operaciones de E/S, que dada la naturaleza del cómputo que se iba a realizar en él iban a ser las más frecuentes. Es por ello que se decide la adquisición de dos disco duros para establecer un RAID1[30] entre ellos.

El RAID 1 crea una copia exacta (o espejo) de los datos en dos o más disco. Esta configuración premia en rendimiento a la lectura de los datos, incrementando el rendimiento como múltiplo lineal del número de copias. Esto implica que un sistema con un RAID1 constituido con 2 discos duros puede estar leyendo simultáneamente dos datos diferentes en dos discos diferentes, consiguiendo duplicar su rendimiento. Por tanto, el tiempo medio de lectura se reduce, ya que los sectores a buscar pueden dividirse entre los discos, bajando el tiempo de búsqueda y subiendo la tasa de transferencia, con el único límite de velocidad soportada por la controladora.

En el servidor, se adquirió una placa base con soporte para RAID1 de forma física (no simulada por software) mediante dos controladoras de disco independiente, una para cada disco (splitting o duplexing).

En las operaciones de escritura, el conjunto RAID1 se comporta como un único disco duro, dato que los datos deben ser escritor en todos los disco que conforman el RAID. Por tanto el rendimiento no mejora. Se consideró esta configuración debido a que se estimó que cada dato era solo escrito una vez (cuando se producía) pero en cambio era requerido de ser leído múltiples veces (en función de los algoritmos de procesamiento que se realizasen). Por tanto, esta configuración que premia la lectura de los datos (frente a la escritura) resultaba la más conveniente.

Además, un sistema RAID1 tiene muchas ventajas a la hora de la administración del sistema. Por ejemplo, en entornos 24/7 como en el que nos encontramos, es posible marcar un disco como “inactivo” para la realización de una copia de seguridad de dicho disco, reconstruyéndose el RAID de forma automática al volver a marcar el disco como activo. Esto supone una gran baza a la hora de obtener instantáneas de los datos. Este mismo sistema de “reconstrucción” funciona igualmente en caso de fallo físico de uno de los discos, con lo cual se mantiene siempre un respaldo del disco duro, lo cual supone un mecanismo de seguridad adicional.

Por último, se descartó añadir al servidor un disco duro sólido (SSD) que pese a ser la alternativa en el mercado más rápida en los tiempos de acceso para lectura, su degradación con el uso suponía un riesgo que difícilmente se podía abordar, además de un coste por GiB de almacenamiento mucho mayor.

4.2 SISTEMA OPERATIVO

Como sistema operativo se instala Ubuntu Server 12.04.03 LTS con el kernel GNU/Linux 3.5.0-40-generic x86_64.

Se opta por este sistema operativo frente a las alternativas planteadas (Debian, CentOS, Ubuntu 13.10) debido a predilecciones personales y familiaridad con el entorno, además de suponer un entorno Libre y disponible para todos los investigadores del proyecto.

Se configura un entorno LAMP[26] para el servidor, que además del procesado de datos servirá algunos servicios y plataformas WEB.

4.2.1 *Configuración adicional del Sistema Operativo: Sistema no-atime*

Para intentar optimizar aún más el acceso a la lectura de los discos duros, se configuran las particiones de los discos duros de forma ‘noatime’ [4].

Por defecto, el sistema de archivos mantiene una variable denominada atime o Access Time para cada fichero, indicando la fecha del último acceso a dicho fichero. Esto supone que para cada lectura del disco, es necesario realizar una escritura (para actualizar dicha fecha). En nuestro sistema esta configuración por defecto nos supone una lacra, pues estamos desaprovechando las ventajas del RAID1, además de estar imponiendo una limitación a la velocidad de lectura.

Para eliminar esta configuración, es necesaria editar la tabla de particiones del servidor, alojada en /etc/fstab y añadir la directiva noatime a las particiones implicadas.

Como experimento de esta configuración, realizamos la siguiente prueba.

Se ejecutará a los 10 minutos del inicio del sistema operativo el siguiente comando:

```
time find /etc -name ".*exec cat '' ''";» /dev/null 2>/dev/null
```

Dicho comando mide el tiempo del sistema necesario para acceder a todos los ficheros alojados en /etc “imprimirlos por pantalla” (aunque se vuelva a salida nula).

Ejecutamos el comando tanto en nuestro sistema por defecto, como en nuestra configuración no-atime:

	Sistema por defecto	Sistema no-atime
Tiempo	0m.2.632s	0m.037s
Ganancia	1	2 049 729

Tabla 4.2

Tiempos de lectura en disco en un sistema por defecto y en un sistema no-atime

Como se puede observar, la mejora es increíblemente significativa. Es por ello que en nuestro entorno preferimos sacrificar la existencia del campo atime, frente a unos accesos a disco muchos más rápidos, que es el fin máximo de este capítulo.

4.2.2 Configuración adicional del sistema operativo: Ficheros temporales y logs en memoria RAM (Descartado)

Una configuración cuyo fin era también la mejora de las operaciones de entrada y salida a disco y que fue planteada (pero desestimada) fue la utilización de particiones de archivos montados en memoria RAM para alojar los ficheros temporales y variables de depuración (o logs del sistema) [38].

Por la naturaleza de estos ficheros, suelen suponer escrituras constante en disco. Se estudió la posibilidad de establecer particiones “simuladas” en la memoria RAM para almacenar esta información. Para configurarlo, hay que editar nuevamente el fichero /etc/fstab y añadir las siguientes particiones:

```
#Archivos temporales y logs alojados en RAM
none /tmp tmpfs size=4%, defaults 0 0
none /var/tmp tmpfs size=3%, defaults 0 0
none /var/log tmpfs size=3%, defaults 0 0
```

Sin embargo, esta configuración supone varios problemas:

- Limita el tamaño de los ficheros temporales y de log al tamaño asignado (el porcentaje de RAM asignada a la partición).

- Reduce la memoria RAM disponible de forma absoluta, pues se reduce la RAM en el total del tamaño de las particiones.
- La información de LOG es perdida en caso de errores que supongan el reinicio o apagado del servidor.

Debido a estas complicaciones, se desestima realizar esta configuración, al menos durante el tiempo en el que el servidor se halle en un entorno aún en desarrollo.

4.3 BASE DE DATOS LOCAL: MYSQL

Aunque más adelante se dedicará un capítulo entero a las bases de datos, este apartado se centrará en las configuraciones realizadas a MySQL^[37] como motor de base de datos a nivel de configuración de servicio. La elección de MySQL como gestor de base de datos para el almacenaje de los datos será también discutida en adelante.

4.3.1 *Deshabilitado de la Query-Caché o caché de consultas*

La caché de consultas o query-caché es un mecanismo proveído por MySQL que se encarga de almacenar el texto de una consulta SELECT junto con el resultado que se envió al cliente. El funcionamiento de este mecanismo es tal que si el motor de base de datos recibe una consulta idéntica posteriormente (en función del tiempo de expiración de la caché) el gestor de base de datos devuelve el resultado en la caché de consultas en lugar de interpretar y ejecutar nuevamente la consulta.

Este tipo de mecanismos de caché son muy empleados (y útiles) en servidores WEB y demás entornos en los que la probabilidad de volver a solicitar unos datos anteriormente pedidos es muy alta. Resultan también muy útiles en entornos donde las tablas no sufren muchos cambios a lo largo del tiempo.

Sin embargo, nuestro sistema es totalmente opuesto a este tipo de naturaleza, pues ofrecemos datos rápidamente cambiados (en tiempo Real) y difícilmente se van a procesar nuevamente las mismas consultas al tratarse de un servidor de cómputo.

Además, este mecanismo supone un cuello de botella a la paralelización de la base de datos, al tener que consultarse “la caché” antes de procesar cualquier consulta. En entornos multihilo (como el que se propone) esto impone que todas las hebras

tiene que acceder al mismo recurso antes de poder procesarse: la caché de consultas.

Es por ello, que se decide deshabilitar la caché de consultas. Para ello es necesario cambiar la variable Global query_cache_type al valor OFF.

4.4 CONFIGURACIÓN DE APACHE

Aunque la tarea principal de nuestro servidor es de cómputo, se va a emplear también para albergar un portal web que publicite el proyecto así como un par de servicios web para suministrar acceso a los datos. Es por ello que configuraremos nuestro servidor con una instalación de APACHE 2.4.6 [7].

Dado que no es un componente vital para el cometido principal del servidor, no se realiza ninguna configuración adicional más que la configuración de dos espacios. Uno el que se encuentra por defecto en el puerto 80 que alojará los portales WEBs, y el otro alojado en el puerto 8080 que se encargará de ofrecer los servicios web.

Para ello, es necesario modificar el fichero que indica los puertos de escucha de APACHE /etc/apache2/ports.conf añadiendo al directivas:

```
listen 80
listen 8080
```

Además, es necesario añadir los ficheros de configuración de los espacios al directorio /etc/apache2/sites-enabled/. En nuestro caso, añadimos el espacio para el portal web en el fichero /etc/apache2/sites-enabled/web-site.conf.

```
#Archivos temporales y logs alojados en RAM
none /tmp tmpfs size=4%, defaults 0 0
none /var/tmp tmpfs size=3%, defaults 0 0
none /var/log tmpfs size=3%, defaults 0 0
```

Y el fichero /etc/apache2/sites-enabled/webservices.conf para el espacio para los servicios web:

```
<VirtualHost *:8080>
ServerAdmin proyectosipesc@gmail.com
Header always append Access-Control-Allow-Origin:
"http://sipesc.ugr.es"
DocumentRoot /var/www/ws
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access_ws.log combined
</VirtualHost>
```

Como en cualquier cambio realizado en el servidor APACHE, es necesario reiniciar el servicio para que se establezcan estos cambios.

5

SIMA: ACTUALIZADOR LOCAL

ÍNDICE DE CAPÍTULO

5.1	Introducción	32
5.2	Descarga de Datos	32
5.2.1	Descarga de Nodos	33
5.2.2	Descarga de Dispositivos y Pasos	34
5.2.3	Paralelizado	36
5.3	Procesado de los Datos	36
5.3.1	Diferentes técnicas de procesado: SPLIT	37
5.3.2	Diferentes técnicas de procesado: JSON	38
5.3.3	Comparativa de diferentes técnicas de procesado	39
5.4	Insercción de los Datos	39
5.4.1	Procesamiento de peticiones en lote	41
5.4.2	Paralelización	42
5.5	Cliente Actualizador DB Local	46
5.5.1	Estructura del módulo	47
5.5.2	Tipos de sincronización	49
5.5.3	Gestión de los nodos	50
5.5.4	Gestión de la ventana de tiempos	50

5.1 INTRODUCCIÓN

En este capítulo se describe el módulo fundamental del sistema SiMa. Este módulo, tiene la tarea principal de encargarse de la adquisición de los datos de cada nodo, tanto de los dispositivos que han sido capturados como de los pasos (ocurrencias) de dichos dispositivos en cada nodo.

Se trata por tanto de una tarea a ser realizada constantemente por nuestro sistema, por lo que se la considera prioritaria. Se aborda esta tarea desde el punto de vista de 3 tareas dependientes entre si.



Figura 5.1
Tareas del sincronizador local de datos

A su vez, estas tres tareas han de ser realizadas para los 3 aspectos que mantenemos sincronizados: los nodos, los dispositivos y los pasos. Estos datos tienen una dependencia tal y como se muestra en la siguiente figura:

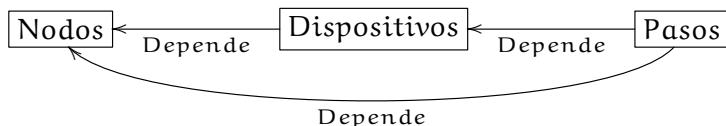


Figura 5.2
Dependencia entre los datos

Por tanto, para los nodos, dispositivos y pasos han de ser descargados, procesados e insertados los datos correspondientes. Además, es un proceso que se debe de dar constantemente en el tiempo.

Como es un proceso complejo, se presentará descompuesto en las distintas partes funcionales, para en la último sección presentar todo el funcionamiento del módulo.

5.2 DESCARGA DE DATOS

Los datos son adquiridos mediante una petición REST a un servidor externo. Los datos no son públicos, por lo que es necesario

emplear unas credenciales de usuario que permiten el acceso a dicha API. Además, los nodos no tienen “temporalidad”, mientras que los dispositivos y pasos necesitan una temporalidad específica. Esto es, para la descarga de los nodos no es necesario introducir ninguna ventana de tiempo (se descarga la información de todos los nodos en el sistema), mientras que para los dispositivos y los pasos es necesario introducir una ventana temporal que indican dos períodos de tiempo que constituyen las fechas umbrales de las que se desea descargar la información.

Por último, los dispositivos y pasos necesitan indicar el nodo del que se desea descargar la información.

Por tanto, presentaremos dos métodos distintos para la descarga, una empleada para los datos atemporales (los nodos) y otra para los datos dependientes y temporales (los pasos y dispositivos)

5.2.1 Descarga de Nodos

Para la descarga de nodos se utiliza un entorno webresource para procesar la petición. Así la función para la descarga de los nodos es la siguiente:

```

1  /**
2   * Función que descarga los nodos registrados en el sistema
3   *
4   * @param responseType clase en la que se desea recibir la respuesta a
5   *                     la petición web
6   */
7  public <T> T get_Nodos(Class<T> responseType) throws
8      UniformInterfaceException {
9      String[] queryParamNames =
10         new String[]{"user", "pass"};
11     String[] queryParamValues =
12         new String[]{_c.get("sc.USER"), _c.get("sc.PASS")};
13
14     return webResource.queryParams(getQueryOrFormParams(queryParamNames,
15                                                 queryParamValues)).get(responseType);
16 }
```

Figura 5.3
Código: Descarga de nodos

En el siguiente capítulo se hablará de la variable `_c` y su importancia como entorno de configuración.

5.2.2 Descarga de Dispositivos y Pasos

La descarga de Dispositivos y Pasos es más elaborada, debido a que debe realizarse para cada nodo y se debe proporcionar una ventana de tiempo válida.

Presentamos a continuación el constructor del cliente encargado de la descarga de los dispositivos:

```
1  /**
2  * Constructor por defecto.
3  *
4  * @param start Fecha de comienzo de los datos
5  * @param end Fecha de fin de los datos
6  */
7  public ClienteDispositivos(String start, String end) {
8      queryParamNames =
9          new String[]{"user", "pass", "start", "end", "inc"};
10
11     queryParamValues =
12         new String[]{_c.get("sc.USER"), _c.get("sc.PASS"), start, end, "
13             true"};
14
15     com.sun.jersey.api.client.config.ClientConfig config =
16         new com.sun.jersey.api.client.config.DefaultClientConfig(); // SSL
17             configuration
18
19     config.getProperties().put(com.sun.jersey.client.urlconnection.
20         HTTPSProperties.PROPERTY_HTTPS_PROPERTIES, new com.sun.jersey.
21             client.urlconnection.HTTPSProperties(getHostnameVerifier(),
22                 getSSLContext()));
23
24     client = Client.create(config);
25
26 }
```

Figura 5.4
Código: Descarga de dispositivos y pasos: constructor

Este código establece los intervalos de tiempo *start* y *end*. Se tratan de variables en formato *string* porque ya se encuentran convertidas al formato requerido por la API REST.

A continuación se establece el nodo para el que se desea descargar la información asociada mediante la función *createWebResource*. Por confidencialidad se ha ocultado la URL del servidor que aloja la API REST.

```

1  /**
2   * Función que establece los parámetros para la conexión HTTP
3   *
4   * @param node Identificador del nodo para el que se va a realiza la
5   * petición HTTP
6   */
7  public void createWebResource(String node) {
8
9      webResource = client.resource("https
10         ://########################################/"
11         + node + "/dispositivos?user=" + queryParamValues[0]
12         + "&pass=" + queryParamValues[1]
13         + "&start=" + queryParamValues[2]
14         + "&end=" + queryParamValues[3]
15         + "&inc=true");
16
17     insertados = 0;
}

```

Figura 5.5

Código: Descarga de dispositivos y pasos: creador del recurso

Ahora sólo es necesario realizar la petición de descarga de los datos mediante la función *get_Dispositivos*:

```

1  /**
2   * Función que realiza la petición de los dispositivos.
3   *
4   * @param <T>
5   * @param responseType
6   * @return La respuesta de la petición en el caso de que haya sido
7   * correcta.
8   * Null en el caso de que se haya producido algún error.
9   */
10  public <T> T get_Dispositivos(Class<T> responseType) {
11
12      try {
13          return webResource.get(responseType);
14      } catch (UniformInterfaceException e) {
15
16          if (e.getResponse().toString().endsWith("returned a response
17          status of 204 No Content")) {
18              Logger.getGlobal().log(Level.FINE,"Error 204 descargando. Se
19              omite.");
20          }
21
22          return null;
23      }
24  }

```

Figura 5.6

Código: Descarga de dispositivos y pasos: realizar descarga

5.2.3 *Paralelizado*

La descarga de los datos es una operación entrada/salida bastante costosa en tiempo, dependiente de la calidad de la conexión y el volumen de los datos. Es por ello que es una sección que invita a ser paralelizada.

Si bien la descarga de los nodos no puede ser paralelizada de ninguna manera (pues se trata de una única petición), no existe problema para realizar la descarga de los dispositivos o pasos de varios nodos al mismo tiempo, o lo que es aquí propuesto, la descarga de los valores mientras se escriben los valores anteriores los valores anteriores. Este mecanismo será descrito en profundidad en el último apartado de este capítulo.

5.3 PROCESADO DE LOS DATOS

Una vez se han descargado los datos, se dispone de una cadena de texto formateada en JSON. Dicha cadena de texto ha de ser procesada, para descomponerla en las unidades básicas de información.

Se propusieron dos métodos para realizar la descomposición. Uno basado en la función SPLIT y otro basado en las clases de procesado de JSON de JAVA.

5.3.1 Diferentes técnicas de procesado: SPLIT

Este método se suponía más rápido, debido a que no es necesario realizar ninguna conversión de `String` a otro formato. En él se utiliza la función `split` para ir partiendo el texto en las diversas partes de información, es decir, en las tuplas y campos.

```
1  /**
2  * Función que procesa mediante SPLIT el resultado de la petición HTTP.
3  *
4  * @param datos
5  * @throws SQLException
6  * @deprecated
7  * @see ClienteDispositivos.procesarDatos(String datos)
8  */
9  public void procesarDatosSplit(String datos) throws SQLException {
10
11     String datosAInsertar = "";
12
13     String[] result = datos.toString().split("}");
14
15     for (int x = 0; x < result.length - 1; x++) {
16
17         String[] result2 = result[x].split(",");
18
19         for (int y = 1; y < result2.length; y++) {
20             String[] result3 = result2[y].split(":");
21
22                 for (int w=1; w<result3.length; w++){
23                     datosAInsertar += result3[1] + ", ";
24                 }
25
26             this.InsertarDatosSync(datosAInsertar.substring(0,
27                                     datosAInsertar.lastIndexOf(",")));
28
29         }
30
31     }
32
33     System.out.println("Dispositivos procesados: " + conta);
34 }
```

Figura 5.7
Código: Procesado: método SPLIT

5.3.2 Diferentes técnicas de procesado: JSON

Este método hace uso de la clase `JSONParser` para crear un objeto que es capaz de navegar de forma nativa por la estructura JSON. En principio, el costo de tener que “construir” la estructura JSON parecía demasiado elevado.

```
1  /**
2   * Función que procesa mediante JSON el resultado de la petición HTTP.
3   *
4   * @param datos La respuesta de la petición HTTP
5   */
6  public void procesarDatos(String datos) {
7      //Preprocesamos los datos, para darle un nombre al array
8      datos = "{\"dispositivos\":\"" + datos + "\"}";
9
10     JSONParser parser = new JSONParser();
11
12     try {
13         JSONObject obj = (JSONObject) parser.parse(datos);
14         JSONArray lista = (JSONArray) obj.get("dispositivos");
15         procesados = lista.size();
16
17         for (int i = 0; i < lista.size(); i++) {
18
19             String a0 = (String) ((JSONObject) lista.get(i)).get("idDispositivo");
20             String a1 = (String) ((JSONObject) lista.get(i)).get("majorDeviceClass");
21             String a2 = (String) ((JSONObject) lista.get(i)).get("minorDeviceClass");
22             String a3 = (String) ((JSONObject) lista.get(i)).get("serviceClass");
23             String a4 = (String) ((JSONObject) lista.get(i)).get("fabricante");
24
25             this.InsertarDatos("\"" + (String) a0 + "\",\"" + a1 + "
26                           ,\"" + a2 + "\",\"" + a3 + "\",\"" + a4 + "\"");
27         }
28     } catch (Exception e) {
29         Logger.getGlobal().log(Level.WARNING, "Fallo en el procesamiento
30                                     de los datos",e);
31     }
32 }
```

Figura 5.8
Código: Procesado: método SPLIT

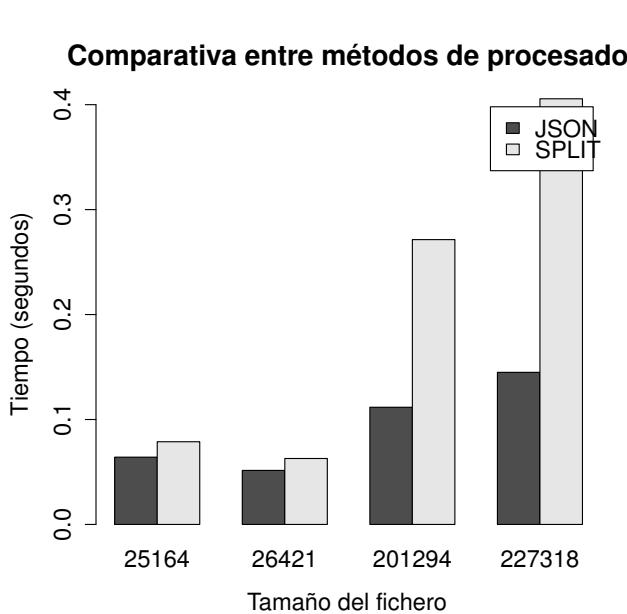


Figura 5.9
Comparativa en tiempos de los distintos métodos de procesado

5.3.3 Comparativa de diferentes técnicas de procesado

Disponiendo de dos métodos de procesamiento de los datos, se realiza un experimento para ver cual de los dos métodos es mucho más eficiente. Se disponen de 4 cadenas de texto de dos tamaños distintos, y se procesan mediante dichos métodos en 3 ocasiones. Nos quedamos con el tiempo promedio de entre todas las mediciones. El resultado de dicho experimento se puede observar en la Figura 5.9.

Se decide por tanto emplear el método que emplea JSON como procesado, pues si bien el tiempo necesario para la conversión es bastante elevado, el tiempo de acceso a los elementos una vez convertido es lo suficientemente ventajoso como para decantarnos por esta opción.

5.4 INSERCIÓN DE LOS DATOS

Para la insercción de los datos se emplea un objeto `Statement` que instancia la conexión a la base de datos. En la siguiente figura puede observarse el código de inserción para el método SPLIT:

```

1  /**
2   * Función que sincroniza de forma síncrona con la Base de Datos Local.
3   * @param datos Petición a ejecutar en la Base de Datos Local.
4   * @throws SQLException Si existe algún error al procesar la petición en
5   *                      la Base de Datos Local.
6   * @deprecated
7   */
8  public void InsertarDatosSync(String datos) throws SQLException {
9      try {
10         String[] comas = datos.split(",");
11         if (comas.length < 5) {
12             datos = datos + ", \"null\" ";
13         }
14
15         Statement st = conexion.crearSt();
16
17         st.executeUpdate("INSERT INTO dispositivo (idDispositivo,
18                         majordeviceclass, minordeviceclass, serviceclass,
19                         fabricante) VALUES (" + datos + " )");
20
21         if ((++insertados) % 100 == 0) {
22             System.err.print("\n+");
23         } else {
24             System.err.print("+");
25         }
26         /*
27          idDispositivo: Cadena tras el hash de la MAC Bluetooth de un
28          dispositivo.
29          majordeviceclass: Cadena con la característica obtenida de la
30          MAC Bluetooth acerca de la procedencia y capacidades del
31          dispositivo.
32          minordeviceclass: Cadena con la característica obtenida de la
33          MAC Bluetooth acerca de la procedencia y capacidades del
34          dispositivo.
35          serviceclass: Cadena con la característica obtenida de la MAC
36          Bluetooth acerca de la procedencia y capacidades del
37          dispositivo.
38          fabricante: Cadena con el nombre del fabricante del dispositivo
39          Bluetooth detectado.
40      */
41      } catch (com.mysql.jdbc.exceptions.
42              MySQLIntegrityConstraintViolationException e) {
43          return;
44      }
45  }

```

Figura 5.10
Código: Versión preliminar método de insertado

Sin embargo aunque este código totalmente funcional, no es todo lo eficiente que puede ser eficiente en el sentido que estudiaremos en los siguientes apartados de este capítulo.

5.4.1 Procesamiento de peticiones en lote

En entornos en los que se suceden las ejecuciones de múltiples ejecuciones, es recomendable la gestión de las peticiones a la base de datos en lote. Esto es, en lugar de realizar las inserciones de 1 en 1, esperar a haber procesado un número n de peticiones antes de enviar la solicitud a la base de datos.

Esto es debido a que el tiempo necesario para el gestor de la base de datos para conseguir el bloqueo exclusivo de la tabla y para posicionar los cabezales es en varios órdenes de magnitud superior la tiempo real de escritura.

Se realiza por tanto un código que se encarga de procesar las peticiones a la base de datos mediante “lotes” de datos.

```

1  /**
2   * Función que añade datos a la cache de peticiones a la base de datos o
3   * bien procesa si se ha superado el tamaño del lote
4   * @param datos Datos a procesar
5   */
6  public void InsertarDatos(String datos) {
7      cache = cache + (cache_size != 0 ? "," : "") + " (" + datos + " ) ";
8      cache_size++;
9
10     if (cache_size >= MAX_CACHE_SIZE) {
11         syncDB();
12     }
}

```

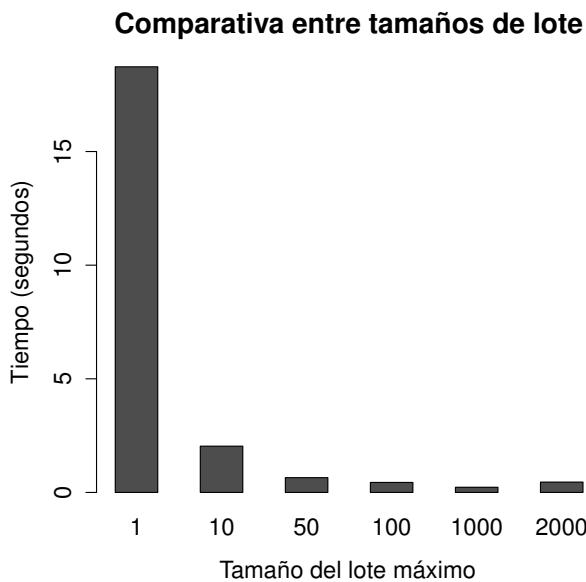
Figura 5.11
Código: Inserción por lotes

La función syncDB es la encargada de enviar la solicitud a la base de datos.

Realizamos un experimento para encontrar un tamaño de caché adecuado. Para ello insertamos en una base de datos de pruebas un fichero procesado con 25164 tuplas con distintos tamaños de caché. La elección de un tamaño máximo de los lotes apropiado para el volumen de datos a insertar es crucial. Si se elige un tamaño de lote muy pequeño, se corre el riesgo de que el tiempo de apertura sea muy superior al de escritura. Sin embargo, también se corre riesgo por elegir un tamaño de lote demasiado grande, ya que si el tamaño del lote es muy grande, no se irán produciendo inserciones en la base de datos hasta haber procesado un número muy alto de tuplas, con lo cual se está “perdiendo” tiempo que se podría aprovechar para insertar.

En la Figura 5.12 se puede observar la reducción de tiempo significativa gracias al empleo del procesado mediante lotes.

Figura 5.12
Comparativa en
tiempos de los
distintos métodos
de procesado



5.4.2 Paralelización

El código de la inserción se ha optimizado de gran manera al realizar el procesamiento por lotes, sin embargo aún puede ser mucho más optimizado. Para ello, haremos uso de mecanismos de paralelización.

De esta forma, no será necesario “esperar” a que la petición a la base de datos haya sido completada para seguir procesando los datos descargados.

Para ello, se implementa una clase heredada de Thread:

```
1  /**
2   * Clase encargada de realizar las peticiones SQL a la Base de Datos
3   * local de forma paralela.
4   */
5  public class threadSyncDB extends Thread {
6
7      /**
8       * Consulta(s) a realizar en la Base de Datos.
9       */
10     private String query;
11
12     /**
13      * Identificador de la hebra.
14      */
15     private int id;
16
17     /**
18      * Variable de conexión con la Base de Datos Local.
19      */
20     private Conectar c;
21
22     /**
23      * Variable de control - Indica el número de intentos de
24      * procesamiento de la petición.
25      */
26     int intentos = 0;
27
28     /**
29      * Variable de control - Indica si la petición ha sido procesada ya
30      * en la Base de Datos Local.
31      */
32     boolean procesada = false;
33
34     /**
35      * Variable de auditoría - Indica el número de elementos que han
36      * sido insertados en la Base de Datos Local.
37      */
38     int insertados = 0;
39
40     /**
41      * Constructor por defecto de la clase.
42      * @param cache Caché de peticiones a realizar en la Base de Datos
43      * Local
44      * @param i Identificador de la hebra.
45      */
46     public threadSyncDB(String cache, int i) {
47         query = cache;
48         id = i;
49     }
50 }
```

Figura 5.13

Código: Clase encargada del procesamiento paralelo de la inserción en la base de datos

Esta clase dispone de dos métodos, el primero de ellos es el que se encarga de forma persistente de realizar la inserción en la base de datos y la finalización de la hebra.

```

1  @Override
2  public void run() {
3      do {
4          try {
5              this.c = new Conectar();
6              Statement st = c.crearSt();
7              insertados = st.executeUpdate("INSERT IGNORE INTO __paso (idNodo,
8                  idDispositivo, tfin, tinicio) VALUES" + query + ";");
9              procesada = true;
10             c.cerrar();
11         } catch (SQLException ex) {
12             procesada = false;
13             intentos++;
14             if (intentos > MAX_ERRORES_PARA_NOTIFICACION) {
15                 Logger.getGlobal().log(Level.WARNING, "Aviso hebra " + this.
16                     getId() + " no ha sincronizado aún con DB. Código " + ex.
17                     getErrorCode() + " Se intentará nuevamente en " +(
18                         TIME_SLEEP_IN_ERROR * intentos * (int) Math.log(Conectar.
19                         current_conections)/1000 + " segundos (Intentos: " +
20                         intentos + ")[Cache:" + cache_size + "][Conexiones: " +
21                         Conectar.current_conections + "]") + "[Hebras: " + l_th.
22                         size() + "]");
23             }
24             try {
25                 sleep(TIME_SLEEP_IN_ERROR * intentos * (int) Math.log(Conectar.
26                     current_conections));
27             } catch (InterruptedException ex1) {
28                 Logger.getGlobal().log(Level.SEVERE, "Error durmiendo hebra " +
29                     this.getId(), ex1);
30             }
31             catch (Exception ex) {
32                 procesada = false;
33                 intentos++;
34                 if (intentos > MAX_ERRORES_PARA_NOTIFICACION) {
35                     Logger.getGlobal().log(Level.WARNING, "Aviso hebra " + this.
36                         getId() + " no ha podido reservar conexión aún a la DB. Se
37                         intentará nuevamente en " + ((int) TIME_SLEEP_IN_ERROR *
38                         intentos * (int) Math.log(Conectar.current_conections)
39                         /1000) + " segundos (Intentos: " + intentos + ")[Cache:" + cache_size + "][Conexiones: " + Conectar.current_conections
40                         + "] " + "[Hebras: " + l_th.size() + "]");
41             }
42             try {
43                 sleep((int) TIME_SLEEP_IN_ERROR * intentos * (int) Math.log(
44                     Conectar.current_conections));
45             } catch (InterruptedException ex1) {
46                 Logger.getGlobal().log(Level.SEVERE, "Error durmiendo hebra " +
                     this.getId(), ex1);
47             }
48         }
49     }
50
51     @Override
52     protected void finalize() throws Throwable {
53         //c.cerrar();
54         conexion.cerrar();
55         super.finalize(); //To change body of generated methods, choose Tools
56             | Templates.
57     }

```

Figura 5.14
Código: Métodos de la clase de inserción paralelo

Caben destacar los mecanismos de persistencia, debido a que al enviarle los lotes de forma paralela, estos tendrán que competir por el recurso: la base de datos.

Por tanto, es posible que no siempre que la hebra quiera acceder a la base de datos pueda hacerlo (pues ya esté el recurso siendo ocupado por otra hebra). Para evitar que las hebras compitan constantemente por el recurso, si una hebra no ha podido acceder al recurso, se dormirá un tiempo que será directamente proporcional al número de intentos y a la cantidad de hebras que estén intentando acceder al recurso. Es interesante “notar” que este sistema perjudica en cierta manera a las hebras más antiguas. Esto es premeditado, ya que priman la importancia de los datos nuevos (actuales) frente a los datos más antiguos.

Con este nuevo mecanismo de paralelismo, las funciones anteriormente mostradas quedan de la siguiente forma:

```

1  public void InsertarDatos(String datos) {
2      cache = cache + (cache_size != 0 ? "," : "") + " (" + datos + " ) ";
3      cache_size++;
4
5      if (cache_size >= MAX_CACHE_SIZE) {
6          syncDB();
7      }
8  }
9
10 public void syncDB() {
11     if (cache_size != 0) {
12         try {
13             l_th.add(new threadSyncDB(cache, l_th.size()));
14             l_th.get(l_th.size() - 1).start();
15         } catch (Exception ex) {
16             Logger.getGlobal().log(Level.SEVERE, "Error creando hebra sincronización con la DB", ex);
17         }
18     }
19     cache_size = 0;
20     cache = "";
21 }
22 }
```

Figura 5.15
Código: Método de inserción paralelo

Sin embargo, es necesario llevar un control del proceso de todas las hebras que tienen que insertar datos. Es por ello que se implementa otra hebra encargada de gestionarlas a todas. El hecho de hacer esto en una hebra independiente en lugar de hacerlo en el propio cliente, es debido a que de esta forma podemos aprovechar los recursos del cliente una vez hayamos terminado de procesar los datos de un nodo y una ventana temporal con

otro nodo. En el siguiente apartado se explicará en detalle este mecanismo.

El código de la clase encargada de controlar el cierre y gestión de las hebras se muestra a continuación:

```

1  public class threadCierre extends Thread {
2
3      private List<threadSyncDB> l = null;
4
5      public threadCierre(List<threadSyncDB> _l) {
6          this.l = _l;
7      }
8
9      @Override
10     public void run() {
11         int insertados = 0;
12         Logger.getGlobal().log(Level.FINE, "Escritura Pasos en BD: " + l.size
13             () + " peticiones");
14         while (!l.isEmpty()) {
15             try {
16                 l.get(0).join();
17                 insertados = insertados + l.get(0).insertados;
18                 l.remove(0);
19             } catch (InterruptedException ex) {
20                 Logger.getGlobal().log(Level.SEVERE, null, ex);
21             }
22         }
23         Logger.getGlobal().log(Level.FINE, "Escritura Pasos en DB OK.
24             Dispositivos insertados/procesados/: " + insertados + "/" +
25             procesados);
26     }
27
28     /**
29      * Función que se encarga de matar a la hebra.
30      * @throws Throwable
31      */
32     @Override
33     protected void finalize() throws Throwable {
34         conexion.cerrar();
35         l.clear();
36         super.finalize();
37     }
38 }
```

Figura 5.16
Código: Clase encargada de la gestión de las hebras de inserción

5.5 CLIENTE ACTUALIZADOR DB LOCAL

En los apartados anteriores, hemos mostrado los diversos mecanismos y optimizaciones llevados a cabo en cada una de las tres tareas principales que tiene que desempeñar este sistema. A pe-

sar de que lo hemos anticipado, no se ha indicado aún como interactúan las diversas tareas entre ellas.

Hemos querido dejar este apartado como el último del capítulo, debido a que conociendo las diversas tareas y sus mecanismos de optimización, se hará más fácil comprender como encajan todas las partes del sistema entre si.

Es necesario entender que este módulo debe de ser capaz de gestionar de forma automática a lo largo del tiempo, que debe realizar las mismas tareas constantemente solicitando datos actuales constantemente.

Además, la eficiencia prima por encima de todo. Este sistema es clave para los sistemas superiores del módulo, por tanto la eficiencia en la adquisición de los datos cercana al tiempo real supone disponer de datos más cercanos al instante actual en los otros módulos.

5.5.1 *Estructura del módulo*

Se espera del módulo que sea adaptativo con los tiempos de actualización, pues nos interesa intentar alcanzar un comportamiento cercano al Tiempo Real. Debido a que todo software es susceptible a fallas, debe de disponer de mecanismos que sean capaces de volver a un estado consistente en el caso de que fallo o reinicio fortuito del sistema.

En la siguiente figura se observa el código resumido que se encarga de la ejecución de este módulo.

```
1  /**
2   * Método principal de la clase.
3   * Ejecuta la actualización de los Nodos, Dispositivos y Pasos en tiempo
4   * real.
5   */
6  @Override
7  public void run() {
8      try {
9          tg = Thread.currentThread().getThreadGroup();
10
11         actualizarNodos();
12         actualizaDesdeFecha();
13
14         //Modo automático
15         do {
16
17             _c.set("data.ultimo", Long.toString(ultimaActualizacion.
18                 getTime()));
19
20             if ((System.currentTimeMillis() - ultimaActualizacion.
21                 getTime()) < TIEMPO_ACTUALIZACIONES_MS) {
22                 long tiempo_espera = (long) ((System.currentTimeMillis()
23                     - ultimaActualizacion.getTime()) *
24                     FACTOR_TIEMPO_ESPERA);
25
26             // .....
27
28             Thread.sleep(TIEMPO_ACTUALIZACIONES_MS - (System.
29                 currentTimeMillis() - ultimaActualizacion.getTime()
30                 ));
31
32             conexion = new Conectar();
33
34             actualizarNodos();
35             actualizaDesdeFecha();
36         } else {
37             long tiempo_espera = (long) (-(System.currentTimeMillis()
38                 - ultimaActualizacion.getTime()) * -
39                     FACTOR_TIEMPO_ESPERA);
40
41             // .....
42
43             actualizarNodos();
44             actualizaDesdeFecha();
45
46         }
47
48         _c.set("data.ultimo", Long.toString(ultimaActualizacion.
49             getTime()));
50         Estadisticas.prime();
51
52     } while (true);
53
54 } catch (ParseException | InterruptedException | SQLException ex) {
55     Logger.getGlobal().log(Level.SEVERE, null, ex);
56 }
57 }
```

Figura 5.17
Código: Estructura del módulo

Se resume por tanto la ejecución de este módulo siguiendo el siguiente proceso:

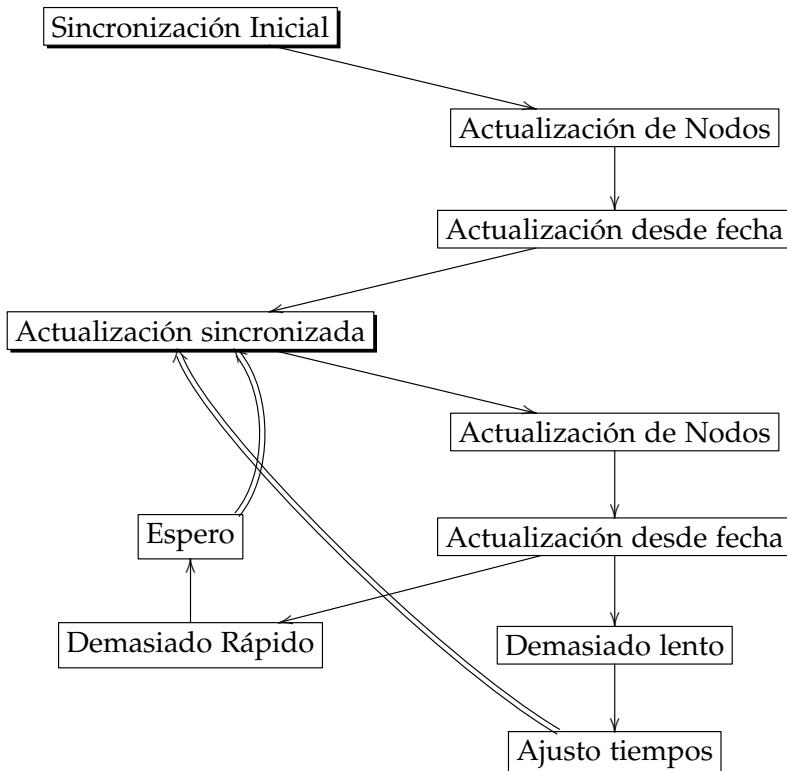


Figura 5.18
Ejecución del módulo

Se distinguen por tanto dos mecanismos de sincronización, una sincronización inicial y una sincronización automática. Y dos tipos de actualización, una actualización de Nodos, y otra actualización desde una fecha concreta.

5.5.2 Tipos de sincronización

La existencia de dos tipos de sincronización obedece a la posibilidad del software de fallar. En caso de fallo, es necesario disponer de un método que sea capaz de reconstruirse conservando la integridad de los datos, consiguiendo los datos que es posible que no se hayan conseguido debido a la falla del software.

Por tanto, la primera actualización a realizar se realiza desde una fecha “segura” donde el software tenga constancia de que estaba en un estado consistente. Este punto de sincronización se puede

observar en la Figura 5.17 en la línea 16. En el siguiente capítulo, descubriremos que misterios se esconden dentro la variable `_c`, de momento lo único que se necesita saber es que almacena el instante de tiempo en el que se va a realizar la siguiente petición, de forma que si el sistema fallase y tuviese que volver a iniciar supiese cual es el instante de tiempo en el que los datos eran consistente.

5.5.3 Gestión de los nodos

Los nodos, como hemos visto, tienen peculiaridades que lo hacen más sencillos de gestionar: no requieren ningún tipo de datos adicional para su descarga. Es por ello que se procesan en una función independiente `actualizarNodos`. En el siguiente código se observa dicha función.

```

1  /**
2   * Actualiza los nodos del sistema.
3   */
4  public static void actualizarNodos() {
5      try {
6          Logger.getGlobal().log(Level.INFO, "Actualizando nodos.");
7          clientNo = new ClienteNodos(conexion);
8          Logger.getGlobal().log(Level.FINE, "Descargando nodos.");
9          response = clientNo.get_Nodos(String.class);
10         Logger.getGlobal().log(Level.FINE, "Procesando nodos.");
11         clientNo.procesarDatos(response.toString());
12         Logger.getGlobal().log(Level.INFO, "Número de nodos actuales: " +
13             clientNo.getHowManyNodos());
14     } catch (SQLException ex) {
15         Logger.getGlobal().log(Level.SEVERE, "Error durante la
16             actualización de nodos.",ex);
17     }
18 }
```

Figura 5.19
Código: Función `actualizarNodos`

Cómo se puede observar, no se hace uso de ningún proceso de paralelización ni gestión concurrente. Esto es debido a la dependencia de los datos, que impide que hasta que esta tarea haya sido completada continuar con las siguientes peticiones.

5.5.4 Gestión de la ventana de tiempos

Al contrario que la sincronización de los nodos, la sincronización de Dispositivos y Pasos, requiere ciertos parámetros en la

API REST. Estos parámetros son el nodo del que queremos adquirir la información y la ventana de tiempo de la cual estamos haciendo la petición.

La ventana de tiempo de las peticiones se gestiona desde la función `actualizaDesdeFecha`, que se presenta a continuación. Presentaremos primero la parte encargada de la actualización de los dispositivos. Como hemos comentado anteriormente, debido al a dependencia de los datos, no es posible comenzar la sincronización de los pasos hasta completar la sincronización de dispositivos:

```

1 /**
2  * Método que actualiza desde la última fecha de actualización
3  * @return 0 - En un futuro se utilizará este parámetro para información
4  *         de la gestión del flujo de trabajo
5  * @throws ParseException
6  * @throws InterruptedException
7  * @throws SQLException
8  */
9 public int actualizaDesdeFecha() throws ParseException,
10    InterruptedException, SQLException {
11
12    Date ahora = Calendar.getInstance().getTime();
13    origen = Calendar.getInstance();
14    origen.setTime(ultimaActualizacion);
15    limite = Calendar.getInstance();
16    limite.setTime(ahora);
17    origen.add(Calendar.HOUR, -VENTANA_HORAS);
18    limite.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
19
20    calendarStart = origen;
21    startDate = calendarStart.getTime();
22    calendarEnd = origen;
23    calendarEnd.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
24    endDate = calendarEnd.getTime();
25
26    do {
27
28        for (int i = 0; i < clientNo.getHowManyNodos(); i =
29            checkChildrens(i)) {
30            System.gc();
31            if (!sigo) {
32                waitForIt();
33            } else {
34                actualizarDispositivos(i,startDate,endDate);
35            }
36        }
37        startDate = endDate;
38        calendarEnd.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
39        endDate = calendarEnd.getTime();
40
41    } while (calendarEnd.before(limite));
42
43    // -----
44    return 0;
45

```

Figura 5.20
Código: Función actualizaDesdeFecha I

Debido a las peculiaridades de JAVA para el tratamiento de fechas, es necesario el empleo de diversas clases para poder trabajar de forma completa con fechas. Por eso se hace uso de objetos tanto tipo `Calendar` como objetos tipos `Date`. Los primeros permiten trabajar con operaciones más complejas como la adición o sustracción de intervalos de tiempo, mientras que el tipo `Date`, permite generar cadenas de textos en diversos formatos aceptados por la API Rest.

Cómo puede observarse, la ventana se ve ampliada en un intervalo, esto es un mecanismo de re-comprobación de los nodos. Es posible, que por motivos externos, se pierda durante algunos minutos la sincronización en tiempo real con los nodos. Ampliando la ventana de petición, una misma tupla es solicitada en varias ocasiones, por lo que si el nodo no ha sido capaz de emitir el dato, este pueda ser adquirido en el sistema.

Se entiende mejor con un ejemplo: Supongamos que la ventana sin "ampliar" va de las 13:30 a las 13:40, siendo las 13:40 la hora actual. Ampliando la ventana, por ejemplo 20 minutos (es decir estableciendo un `NUMERO_MINUTOS_PETICION = 20` se realizaría una petición en una ventana comprendida entre las 13:10 y las 14:00. El ampliar también a tiempo futuro, es un mecanismo de "sobreestimación" que no supone ninguna complicación en término general, pero que en caso de un comportamiento anómalo puede permitir traer nodos aunque la sincronización haya empezado "más tarde". De esta forma, por ejemplo, un dispositivo localizado a las 13:35 tendría tres oportunidades de ser añadido al sistema. Entendemos que una desconexión de más de 20 minutos de un nodo, puede suponer un problema que necesite gestión urgente.

En el código se presentan dos mecanismos para la sincronización, el primero de ellos es la función `checkChildrens` que se presenta a continuación:

```

1  /**
2   * Función de concurrencia: Comprueba el número de hebras hijas que
3   * tiene el
4   * proceso PADRE, y en caso de ser un número aceptable, incrementa el
5   * contador.
6   *
7   * @param i Parámetro que del contador actual
8   * @return i si el número de hebras vivas está por encima de lo
9   * aceptable,
10  * i++ si el número de hebras vivas no supera el máximo permitido.
11  */
12 private static int checkChildrens(int i) {
13     if(clientPa != null ) {
14         if (clientPa.l_th.size() > MAX_HEBRAS_EN_COLA_PARA_DESCARGAR) {
15             sigo = false;
16         } else {
17             i++;
18             sigo = true;
19         }
20     }
21     return i;
22 } else{
23     sigo = true;
24     i++;
25     return i;
26 }
27 }
```

Figura 5.21
Código: Función checkChildrens

La función `checkChildrens` se encarga por tanto de controlar que el número de hebras de inserción pendiente no supere un máximo establecido. De esta forma, evitamos seguir procesando datos cuando el número de hebras pendientes es demasiado elevado.

La función que se encarga de esperar ese tiempo es la función `waitForIt`, que presentamos a continuación:

```

1  /**
2   * Función de concurrencia: Duerme el proceso padre un cierto tiempo,
3   * mientras espera que se procesen los hijos.
4   *
5   * @throws InterruptedException Si no se puede dormir el proceso padre
6   */
7 private static void waitforIt() throws InterruptedException {
8     long t = TIME_BASE_ESPERA * (clientPa.l_th.size() + 1) + (tg.
9         activeCount() * INCREMENTO_TIME_POR_HEBRA);
10    contadorRepeticion++;
11    // -----
12    Thread.sleep(t);
13 }
```

Figura 5.22
Código: Función whaitForIt

Dicha función se encarga por tanto de dormir al cliente de sincronización un tiempo que es directamente proporcional a la cantidad de hebras que hay pendientes de procesar.

El código que hemos mostrado, se encarga sólo de actualizar los Dispositivos, a continuación se presenta el código que se encarga de la actualización de los pasos que como se puede observar es análogo al de los dispositivos:

```

1 /**
2  * Método que actualiza desde la última fecha de actualización
3  * @return 0 - En un futuro se utilizará este parámetro para información
4  *         de la gestión del flujo de trabajo
5  * @throws ParseException
6  * @throws InterruptedException
7  * @throws SQLException
8  *
9 */
10 public int actualizaDesdeFecha() throws ParseException,
11     InterruptedException, SQLException {
12
13 // -----
14
15     origen = Calendar.getInstance();
16     origen.setTime(ultimaActualizacion);
17     limite = Calendar.getInstance();
18     limite.setTime(ahora);
19     origen.add(Calendar.HOUR, -VENTANA_HORAS);
20     limite.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
21     calendarStart = origen;
22     startDate = calendarStart.getTime();
23     calendarEnd = origen;
24     calendarEnd.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
25     endDate = calendarEnd.getTime();
26
27     do {
28         for (int i = 0; i < clientNo.getHowManyNodos(); i =
29             checkChildrens(i)) {
30             System.gc();
31             if (!sigo) {
32                 waitForIt();
33             } else {
34                 actualizarPasos(i,startDate, endDate);
35             }
36         }
37         startDate = endDate;
38         calendarEnd.add(Calendar.MINUTE, NUMERO_MINUTOS_PETICION);
39         endDate = calendarEnd.getTime();
40
41     } while (calendarEnd.before(limite));
42
43     //Actualizamos la fechad de la última actualización
44     ultimaActualizacion = ahora;
45
46     return 0;

```

Figura 5.23
Código: Función actualizaDesdeFecha II

Este mecanismo permite descargar y preparar hebras de insertando en la base de datos a pesar de que haya peticiones en la base de datos pendiente de insertar, aunque sin superar un número máximo que pueda hacer peligrar la base de datos.

CONCLUSIONES

Terminamos el capítulo haciendo un resumen del módulo presentado. Este módulo es el encargado de la recopilación de los datos de los nodos, lo cual lo convierte en un sistema prioritario dentro de nuestro proyecto. La optimización se ha cuidado con mimo, debido a que su eficiencia marcará los tiempos del resto del sistema. Se ha presentado el sistema con “carga segura” en caso de error. Se ha mostrado también el sistema paralelo basado en colas para la inserción en la base de datos de los nodos, que también permite la descarga y procesado de los datos mientras se realizan las escritura en el disco.

6

SIMA: GESTIÓN DE LA CONFIGURACIÓN

ÍNDICE DE CAPÍTULO

6.1	Uso de ficheros properties para la configuración	60
6.2	Sistema de doble fichero de configuración	62
6.2.1	Ficheros properties encriptados	63
6.3	Uso de módulo de configuración	64
6.4	Parámetros de configuración del sistema	65
6.4.1	Modo Depuración	65
6.4.2	Variables de conexión a la base de datos Local	65
6.4.3	Variabes de optimización de las consultas a la base de datos Local	65
6.4.4	Variabes de optimización para la concurrencia de las peticiones a la base de datos Local	66
6.4.5	Directorio de configuración y almacén de credenciales	66
6.4.6	Variabes de configuración de Fusion Tables	66
6.4.7	Credenciales para el acceso a la información de los nodos	68
6.4.8	Parámetros de configuración del cliente de twitter	68
6.4.9	Constantes geográficas	69

En el anterior capítulo hemos utilizado en varias ocasiones la variable `_c` sin entrar en más detalles sobre su naturaleza. En este capítulo abordaremos el sistema de configuración del sistema que se hace accesible mediante dicha variable. De esta manera, empleamos la variable `_c` en todos los módulos para poder acceder a la configuración del sistema.

Para hacer uso del entorno de configuración, sólo es necesario instanciar un objeto de tipo Config en el módulo donde se desee iniciar. Se recomienda hacerlo empleando una variable static para mejorar la eficiencia e integridad del fichero de configuración.

```
1 //Variables de configuración  
2 static Config _c = new Config();
```

Figura 6.1
Código: Inicialización del entorno: Configuración

6.1 USO DE FICHEROS PROPERTIES PARA LA CONFIGURACIÓN

Java dispone de la clase properties para gestionar ficheros de configuración, la cual resulta muy útil para almacenar todos aquellas constantes que pueden ser empleadas dentro del código. Las properties pueden ser almacenadas o cargadas en un flujo (por ejemplo un fichero) para permitir conservar información entre ejecuciones. Esta clase proporciona un sistema {clave,valor} de rápido acceso mediante una tabla hash, así como métodos para la lectura y escritura en fichero properties.

Nuestro módulo de configuración, obedece a dos necesidades distintas. Por un lado, dotar la software de la capacidad de ser modificado sin necesidad de recompilar todo el software. Por otro lado ofrece la capacidad para almacenar el estado consistente del sistema, para en caso de error ser capaz de reconstruirse manteniendo siempre la consistencia e integridad de los datos.

Presentamos por tanto la clase config que codifica nuestro módulo de configuración:

```
1 /**
2  * Clase manejadora de la configuración de la aplicación
3  * Emplea dos archivos Properties de configuración, uno global en el propio
4  * JAR (sólo lectura) y uno local en el home del usuario (permite
5  * sobreescribir las propiedades)
6  * @author Antonio Fernández Ares (antares.es@gmail.com)
7  */
8 public class Config {
9
10    /*
11     * Fichero de propiedades global
12     */
13    public Properties _c_global = new Properties();
14
15    /*
16     * Ficheros de propiedades local
17     */
18    public Properties _c_local = new Properties();
19
20    /*
21     * Método que carga la configuración de los ficheros properties
22     */
23    private void cargarConfiguracion() {
24        try {
25            _c_global.load(Config.class.getResourceAsStream("config.
26                properties"));
27            File fichero = new File(System.getProperty("user.home") + "/" +
28                _c_global.getProperty("directorio_configuracion") + "/"
29                "config.properties");
30            _c_local.load(new FileInputStream(fichero));
31        } catch (Exception ex) {
32            Logger.getGlobal().log(Level.SEVERE,"no se ha podido cargar el
33                fichero de Configuración", ex);
34        }
35    }
36
37    /**
38     * Constructor de la clase
39     * @
40     */
41    public Config() {
42        this.cargarConfiguracion();
43    }
44}
```

Figura 6.2
Código: Clase encargada de la gestión de la configuración I

Si observamos el código, podemos observar que se hace uso de dos ficheros properties para la gestión de la configuración. En la siguiente sección veremos el porque de este sistema de doble fichero.

6.2 SISTEMA DE DOBLE FICHERO DE CONFIGURACIÓN

Cuando se trabaja con un fichero properties este puede estar alojado en dos zonas distintas: o bien como recursos (dentro del JAR) o bien como fichero externo. Ambos métodos tienen sus ventajas e inconvenientes. Si bien ambos son accesibles y modificables por el usuario, en tiempo de ejecución modificar un fichero alojado en disco es más sencillo que modificar un fichero alojado como recurso. A nivel de seguridad, ambos son accesibles y modificables por el usuario, por lo que no existen diferencias en este aspecto.

A nivel de “localización” es mucho más sencillo acceder a un recurso (pues sólo se requiere el nombre) frente al acceso a un archivo (que se requiere el nombre y la ruta del archivo).

Nos decantamos por la implementación de un sistema de configuración de doble fichero para poder disponer de dos entornos de configuración:

- Global: Un fichero properties alojado dentro del propio JAR con toda la información de configuración. En tiempo de ejecución, tiene sólo permisos de lectura.
- Local: Un fichero properties alojado en disco en el directorio del usuario, con permisos de Lectura y Escritura en tiempo de ejecución.

De esta forma, podemos disponer de un doble mecanismo para la gestión de la configuración: un entorno de sólo lectura (con la configuración básica) y otro entorno con permisos de lectura/escritura (para almacenar la el estado consistente de los datos, como se vio en la sección anterior).

A la hora de solicitar un dato, los datos se consideran con la jerarquía mostrada en la Figura 10.1. Esta jerarquía es debido a que el entorno local es posible que sea actualizado, por tanto si existe un valor almacenado en él este tiene prioridad frente al valor almacenado el en entorno GLOBAL.

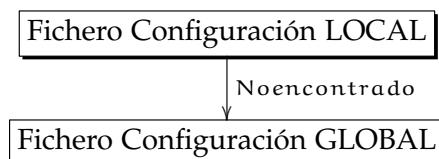


Figura 6.3
Configuración: Jerarquía de búsqueda

Por tanto, al solicitar un valor se debe buscar primero en el fichero properties local. Se presentan a continuación los métodos de *get* y *set* del módulo de configuración, encargados de leer un valor respecto su clave y de escribir un valor a una clave determinada, respectivamente.

```

1  /**
2   * Devuelve el valor de una determinada clave
3   * @param key La clave a buscar
4   * @return Busca la clave primero en la configuración local, si no está
5   * especificada, en la configuración global, si no existe, se
6   * devuelve null
7   */
8  public String get(String key) {
9      if (_c_local.getProperty(key) != null) {
10         return _c_local.getProperty(key);
11     } else {
12         return _c_global.getProperty(key);
13     }
14
15     /**
16      * Inserta o modifica el valor de una determinada clave en la
17      * configuración local
18      * @param key La clave bajo la que se almacenará el valor
19      * @param valor El valor que se asociará a la clave
20      */
21     public void set(String key, String valor) {
22         try {
23             _c_local.setProperty(key, valor);
24             _c_local.store(new FileOutputStream(System.getProperty("user.
25                 home") + "/" + _c_global.getProperty(""
26                 directorio_configuracion") + "/config.properties"), null);
27         } catch (Exception ex) {
28             Logger.getGlobal().log(Level.SEVERE, "No se ha podido cargar el
29                 fichero de configuración", ex);
30         }
31     }
32 }
```

Figura 6.4
Código: Clase encargada de la gestión de la configuración II

6.2.1 *Ficheros properties encriptados*

Durante el diseño del módulo de configuración, se estudió la vulnerabilidad de los ficheros properties, pues al encontrarse en texto plano, eran legibles por el propio usuario, lo cual suponía una brecha de seguridad. Es por ello que se planteó la posibilidad de encriptar el fichero properties de la configuración Global, con el fin de salvaguardar esta información.

Para ello se hace uso de la clase *Encryptableproperties* para crear y encriptar el fichero properties global. Sin embargo, por

comodidad en la fase de desarrollo del proyecto, esta es una funcionalidad que aún no ha sido implementada, a pesar de haber sido diseñado. Se considera una funcionalidad que será implementada en las fase final de desarrollo del proyecto, cuando ya no sea requerido realizar cambios de configuración en el entorno.

6.3 USO DE MÓDULO DE CONFIGURACIÓN

Como hemos visto, para invocar el módulo de configuración sólo es necesario instanciar un objeto (recomendable que se estático) como hemos visto en la Figura 6.1. Una vez instanciado el objeto se pueden emplear los métodos presentados en la Figura 6.4 así como los métodos adicionales de conversión que se presentan a continuación:

```
1  /**
2   * Devuelve el valor de una determinada clave en formato entero
3   * @param key La clave a buscar
4   * @return Busca la clave primero en la configuración local, si no está
5   *         especificada, en la configuración global, si no existe, se
6   *         devuelve null
7   */
8  public int getInt(String key) {
9      return Integer.parseInt(this.get(key));
10 }
11
12 /**
13  * Devuelve el valor de una determinada clave en formato long
14  * @param key La clave a buscar
15  * @return Busca la clave primero en la configuración local, si no está
16  *         especificada, en la configuración global, si no existe, se
17  *         devuelve null
18  */
19 public long getLong(String key) {
20     return Long.parseLong(this.get(key));
21 }
22
23 /**
24  * Devuelve el valor de una determinada clave en formato booleano
25  * @param key La clave a buscar
26  * @return Busca la clave primero en la configuración local, si no está
   *         especificada, en la configuración global, si no existe, se
   *         devuelve null
   */
   public boolean getBool (String key){
       return Boolean.parseBoolean(this.get(key));
   }
```

Figura 6.5
Código: Clase encargada de la gestión de la configuración III

6.4 PARÁMETROS DE CONFIGURACIÓN DEL SISTEMA

A modo de recopilatorio, se presentan a continuación los parámetros de configuración del sistema modificables mediante este módulo y que repercuten en la ejecución y comportamiento de todo el sistema SiMA.

Muchos parámetros, al pertenecer a sistemas que aún no han sido descritos, no se comprenderán en este apartado. Invitamos al lector a volver a esta sección después de la lectura de cada capítulo para observar los parámetros de configuración de cada módulo.

6.4.1 *Modo Depuración*

debug (true | false) Indica si está activo o no el modo de depuración del sistema.

6.4.2 *Variables de conexión a la base de datos Local*

Estos parámetros configuran el acceso a la base de datos local.

db.basedatos Nombre de la base de datos donde se almacenan los datos locales.

db.usuario Usuario en la base de datos donde se almacenan los datos locales.

db.contraseña Contraseña del usuario anteriormente especificado en la base de datos donde se almacenan los datos locales.

db.host Servidor donde se aloja la base de datos donde se almacenan los datos locales.

6.4.3 *Variables de optimización de las consultas a la base de datos Local*

Variables que controlan la optimización mediante el procesamiento por lotes de las peticiones de inserción a la base datos, es decir, el número de tuplas que son insertadas en la misma petición para cada uno de los tipos de datos a insertar.

db.nodo.max_cache_size Tamaño máximo del lote en las inserciones de nodos en la base de datos.

db.dispositivo.max_cache_size Tamaño máximo del lote en las inserciones de dispositivos en la base de datos.

db.paso.max_cache_size Tamaño máximo del lote en las inserciones de pasos en la base de datos.

6.4.4 Variables de optimización para la concurrencia de las peticiones a la base de datos Local

Variables que controlan la optimización mediante el sistema de colas para las peticiones de insercción a la base de datos.

db.dispositivo.time_sleep_in_error Tiempo en ms que debe dormir la hebra de inserción en caso de no haber podido acceder al recurso en la inserción de pasos.

db.dispositivo.max_hebras_activas_simultaneas Número de hebras que pueden estar accediendo de forma simultánea al recurso en la inserción de pasos.

db.dispositivo.max_errores_para_notificacion Número de intentos de acceso al recurso a partir del cual se notifica de acceso erróneo al sistema en la inserción de pasos.

db.paso.time_sleep_in_error Tiempo en ms que debe dormir la hebra de inserción en caso de no haber podido acceder al recurso en la inserción de pasos.

db.paso.max_hebras_activas_simultaneas Número de hebras que pueden estar accediendo de forma simultánea al recurso en la inserción de pasos.

db.paso.max_errores_para_notificacion Número de intentos de acceso al recurso a partir del cual se notifica de acceso erróneo al sistema en la inserción de pasos.

6.4.5 Directorio de configuración y almacén de credenciales

directorio_configuracion Ruta al directorio de configuración, donde se almacenan entre otros el fichero properties local y las credenciales oAuth.

6.4.6 Variables de configuración de Fusion Tables

Variables que controlan el módulo de sincronización en la nube mediante el servicio Google Fusion Tables.

Rutas e identificadores de tablas

Identifican las rutas para la peticiones así como los identificadores de las distintas tablas almacenadas en el servicio.

ft.service_url Dirección para las solicitudes a la API REST de Google Fusion Tables.

ft.application_name Nombre de la aplicación con acceso al servicio Google Fusion Tables.

ft.data_store_file Nombre del fichero donde se almacenan las credenciales de acceso mediante oAuth.

ft.pasospordia.id Identificador de la tabla encargada de almacenar el resumen de los datos de pasos de vehículos agrupados por día.

ft.pasosporhora.id Identificador de la tabla encargada de almacenar el resumen de los datos de pasos de vehículos agrupados por horas.

ft.trazasporhoras.id Identificador de la tabla encargada de almacenar el resumen de los datos de trazas de vehículos agrupados por horas.

ft.nodos.id Identificador de la tabla encargada de almacenar la información de los nodos.

Configuración de la sincronización temporal con el servicio Google Fusion Tables

ft.tiempo_resincronizar Tiempo (en milisegundos) en la que se resincroniza la información en el Servicio Google Fusion Tables.

ft.tiempo_espera Tiempo de espera (en milisegundos) en caso de haber superado la cuota de peticiones realizadas al Servicio Google Fusion Tables antes de volver a intentar enviar un petición.

ft.periodo_actualizacion Tiempo (en milisegundos) que indica cada cuanto tiempo se suben valores actualizados al Servicio Google Fusion Tables.

ft.tiempo_espera_error_ms Tiempo (en milisegundos) de espera en caso de error procesando la petición al Servicio Google Fusion Tables antes de volver a intentar el envío.

ft.tiempo_espera_entre_peticiones_ms Tiempo (en milisegundos) de espera entre envíos al Servicio de Google Fusion Tables.

ft.insert_cache_size Número de tuplas que son enviadas de forma simultaneas en cada petición de inserción al Servicio Google Fusion Tables.

ft.primera_vez (true | false) Variable que indica si es la primera vez que se realiza la subida de datos al Servicio Google Fusion Tables.

ft.vectores_sincronizacion_nodos Cada cuantas sincronizaciones se actualiza la información de los nodos en el Servicio Google Fusion Tables.

ft.tiempo.esperasubida.dormir Tiempo (en milisegundos) que debe dormir una hebra cuando intenta acceder al recurso y este se encuentra ocupado.

Configuración de la sincronización temporal con el servicio Google Fusion Tables para el sistema de predicción

ft.prediccion_tiempo_espera Tiempo de espera antes de comenzar la subida de la predicción en el Servicio Google Fusion Tables en la primera ejecución.

ft.prediccion_periodo_actualizacion Periodo de actualización (en milisegundos) que indica cada cuanto tiempo se suben nuevos valores de predicción al Servicio Google Fusion Tables.

ft.vectores_prediccion_pasos_por_horas Cada cuantas “subidas” al Servicio Google Fusion Tables se actualiza el agente de predicción.

6.4.7 Credenciales para el acceso a la información de los nodos

sc.user Usuario para el acceso a la información de los nodos.

sc.pass Contraseña del usuario para el acceso a la información de los nodos.

sc.nodo.base_uri Dirección web de acceso a la API REST que provee de información a los nodos.

6.4.8 Parámetros de configuración del cliente de twitter

twitter.url Dirección del bot encargado de la publicación en twitter.

6.4.9 *Constantes geográficas*

tierra.radio Constante que indica el radio de la tierra en metros (empleado para las operaciones de conversión entre coordenadas geográficas)

7

SIMA: MECANISMOS DE DEPURACIÓN

ÍNDICE DE CAPÍTULO

7.1	Entorno de depuración: Empleo de objetos Logger	72
7.1.1	Formato personalizado de fichero de depuración.	74
7.1.2	Ejemplo de uso	75
7.2	Mediciones de tiempo	76
7.2.1	Ejemplo de uso	77
7.3	Formatos de fecha y números flotantes	77
7.3.1	Ejemplo de uso	78

En un sistema complejo, proveer los mecanismos de depuración adecuados es de vital importancia. Aún más, cuando se trata de un sistema que procesa los datos Tiempo Real, en el que la reproducción de errores es prácticamente imposible. Además, su naturaleza 24/7 obliga a capturar de forma eficiente los errores, pues no se dispone de un operador que compruebe constantemente la correcta ejecución del entorno.

En este capítulo se describe el módulo de depuración, que ha sido de vital importancia para la depuración y correcto funcionamiento de todo el sistema. Sin el diseño y uso de un sistema de depuración la detección de anomalías hubiese sido imposible, comprometiendo la integridad de los datos ofrecidos por el sistema, lo cual hubiese supuesto un auténtico fracaso en el procesado de los datos.

Al igual que con el módulo de configuración, el modo de acceso al módulo de depuración se realiza mediante la instanciación de un objeto estático, tal y como se muestra en la siguiente figura:

```
1 //Variables de depuración  
2 static Debug _c = new Debug();
```

Figura 7.1
Código: Inicialización del entorno: Depuración

7.1 ENTORNO DE DEPURACIÓN: EMPLEO DE OBJETOS LOG-GER

Para la gestión de la depuración se ha hecho uso del objeto global Logger, que permite gestionar la salida a ficheros, provee de diversos niveles de depuración y aunque no es la alternativa más avanzada, es lo suficientemente potente y simple como para poder modificarla, ampliarla y mejorarla a nuestro sistema.

Se provee por tanto de una clase llamada Debug que hace sirve para establecer la configuración del sistema de Log del sistema. En la siguiente figura, se observa el código que genera la salida de log del sistema.

```
1 public void generarConfiguracion(){
2     try {
3         //Creamos el directorio de logs si no existe
4         File dir_logs = new File("logs");
5         if (!dir_logs.exists()) {
6             dir_logs.mkdir();
7         }
8
9         Logger.getAnonymousLogger().setLevel(Level.ALL);
10        Logger.getGlobal().setLevel(Level.ALL);
11
12        //Fichero de log global
13        FileHandler f_global = new FileHandler("logs/log", 314572800, 6, true)
14            ;
15        f_global.setLevel(Level.ALL);
16        f_global.setFormatter(new formatoLog());
17
18        Logger.getGlobal().addHandler(f_global);
19        Logger.getGlobal().removeHandler(new ConsoleHandler());
20
21        //Fichero de log global en XML
22        FileHandler f_global_XML = new FileHandler("logs/xml", 314572800, 6,
23            true);
24        f_global_XML.setLevel(Level.WARNING);
25        f_global_XML.setFormatter(new XMLFormatter());
26
27        Logger.getGlobal().addHandler(f_global_XML);
28
29        //Fichero dedicado para los errores
30        FileHandler f_global_error = new FileHandler("logs/error", 314572800,
31            3, true);
32        f_global_error.setLevel(Level.SEVERE);
33        f_global_error.setFormatter(new formatoLog());
34
35        Logger.getGlobal().addHandler(f_global_error);
36
37        //Fichero dedicado para los errores
38        FileHandler f_info = new FileHandler("logs/info", 314572800, 3, true);
39        f_info.setLevel(Level.INFO);
40        f_info.setFormatter(new formatoLog());
41
42        Logger.getGlobal().addHandler(f_info);
43
44    } catch (IOException | SecurityException ex) {
45        Logger.getGlobal().log(Level.SEVERE, null, ex);
46    }
}
```

Figura 7.2

Código: Inicialización del entorno: Configuración del sistema de Depuración

Como se observa, se generan 4 ficheros de depuración que se describen a continuación:

logs/info Fichero que almacena la información de información del sistema en un formato personalizado.

logs/xml Fichero que almacena los mensajes con nivel superior a ADVERTENCIA en formato XML con información avanzada de depuración.

logs/error Fichero que almacena los mensajes con nivel superior a ERROR en formato personalizado.

logs/log Fichero que almacena TODOS los mensajes de depuración del sistema en un formato personalizado.

La ventaja del uso de Logger es que provee mecanismos para almacenar los ficheros de log con un tamaño máximo y un número máximo de ficheros por cada log. Así en nuestro caso, cada log puede emplear hasta 6 ficheros de 314572800 bytes (300MB por fichero). En caso de llenar los 6 ficheros, se elimina automáticamente los datos más antiguos.

7.1.1 Formato personalizado de fichero de depuración.

Hemos comentado que hemos empleado (salvo en la salida ampliada XML) un formato personalizado para la salida de depuración. Si bien por defecto la clase Logger provee de varios formatos para la salida de datos, consideramos que ninguno de ellos ofrecía ni el formateado ni la información que creíamos necesaria para nuestro sistema.

Además, como se verá en el Capítulo ?? donde se presenta la herramienta del Visor de Eventos, haber desarrollado nuestro propio formato para la salida de depuración nos permite implementar software que sea capaz de trabajar con esa salida para presentar de forma más gráfica la ejecución de nuestro sistema.

Para definir el formato personalizado de depuración, se ha implementando una clase que extiende las funcionalidades de la clase `java.util.logging.SimpleFormatter` y se ha sobreescrito el método `format` como se puede observar en la siguiente figura.

```

1 /**
2  * Clase que indica el formato de salida del fichero de log
3  * Basado en la clase java.util.logging.SimpleFormatter
4  * @author antares
5  */
6 public class formatoLog extends java.util.logging.SimpleFormatter {
7
8     private final Date dat = new Date();
9
10    @Override
11    public synchronized String format(LogRecord record) {
12        dat.setTime(record.getMillis());
13        return Debug.sdf.format(dat) + "\t" + record.getLevel().getName() + \
14            "\t" + record.getSourceClassName() + "\t" + record.
15            getSourceMethodName() + "\t" + record.getMessage() + "\n";
16    }
17 }
```

Figura 7.3
Código: Formato personalizado de depuración

Así se ha añadido una marca temporal a cada registro del log pues creemos que en un sistema como el nuestro saber la fecha de ocurrencia de cada registro es importante. Además se hace uso de una estructura tabular muy sencilla, frente a alternativas de formatos más complejos (como XML o JSON). Precisamente esta simpleza, es la que permite trabajar de forma tan rápida y eficiente del sistema de log.

7.1.2 Ejemplo de uso

Para emplear el sistema de depuración sólo es necesario emplear la variable global Logger habiendo previamente instanciado un objeto de tipo Debug (que puede hacerse una única vez al comienzo de la ejecución del sistema). En la siguiente figura, se muestran distintos tipos de mensajes que pueden ser enviados.

```

1 //Mensaje de información
2 Logger.getGlobal().info("Esto es un mensaje de información");
3
4 //Mensaje de error
5 Logger.getGlobal().severe("Esto es un mensaje de error");
6
7 //Mensaje de advertencia
8 Logger.getGlobal().warning("Esto es un mensaje de advertencia");
9
10 //Mensaje de que todo va bien
11 Logger.getGlobal().fine("Esto es un mensaje de todo va bien");
```

Figura 7.4
Código: Uso del sistema de depuración

De forma automática, el sistema de depuración escribirá en el fichero (o ficheros) que corresponda el mensaje correspondiente, además de la información adicional que hemos configurado con nuestro formato local.

7.2 MEDICIONES DE TIEMPO

En ocasiones, la depuración no consiste solamente en la detección y notificación de errores. En un sistema con el que estamos trabajando, en el que se ha mirado al detalle la optimización y eficiencia del código, proveer de un mecanismo de depuración para la medición de tiempo supone una gran herramienta.

Es por ello que el módulo de depuración implementado dispone de funciones para la medición de tiempo.

```
1  /**
2   * Variables para medida de tiempo
3   */
4  static long t_start, time;
5  /**
6
7  /**
8   * Función de depuración: Almacena el tiempo actual para realizar una
9     medición posterior
10  *
11  */
12 public void timeCheck() {
13     t_start = System.currentTimeMillis();
14 }
15 /**
16   * Función de depuración: Imprime el tiempo transcurrido desde la última
17     marca de tiempo
18  *
19  * @param reset Booleano que indica si se tiene que establecer una
20    nueva marca de tiempo tras realizar la medición
21  * @return Cadena de texto con la medición de tiempo
22  */
23 @Deprecated
24 public String timeDisplay(boolean reset) {
25     time = System.currentTimeMillis() - t_start;
26     String t = "(" + df.format(time / 1000.0 / 60.0) + "min)";
27     if (reset) {
28         t_start = System.currentTimeMillis();
29     }
30     return t;
31 }
```

Figura 7.5
Código: Sistema de depuración: mediciones de tiempo

Si bien es un sistema bastante sencillo que sólo permite establecer un sólo temporizador al mismo tiempo, no se consideró necesario realizar un sistema más complicado.

7.2.1 Ejemplo de uso

A continuación se muestra un código de ejemplo donde se realiza la medición de una funcionalidad concreta.

```
1 // ----- //
2 _d.timeCheck();
3     ClienteDispositivos clientDi;
4     clientDi = new ClienteDispositivos( String.valueOf(start.getTime()
5         ), String.valueOf(end.getTime()));
6     clientDi.setLabel(label);
7     clientDi.createWebResource(clientNo.getNodo(i));
8     response = clientDi.get_Dispositivos(String.class);
9     Logger.getGlobal().fine("Descarga OK" + _d.timeDisplay(true) +
10         " Procesando ");
11    clientDi.setConexion(conexion);
12    if (response != null) {
13        clientDi.procesarDatos(response.toString());
14    }
15 Logger.getGlobal().fine("Procesado OK " + clientDi.getProcesados() + _d.
16     timeDisplay(true));
17     clientDi.close();
18 // ----- //
```

Figura 7.6

Código: Sistema de depuración: mediciones de tiempo (ejemplo de uso)

7.3 FORMATOS DE FECHA Y NÚMEROS FLOTANTES

En nuestro sistema se trabaja constantemente con fechas y horas, sin embargo, existen innumerables representaciones de estos tipos de variables y muchos de ellos no son fácilmente interpretables por los seres humano, por lo que proveer un mecanismo de depuración que nos permita traducir a un formato legible las fechas y horas es una gran herramienta.

De igual manera, los números flotantes en ocasiones pueden ser demasiado grandes, dificultando su interpretación de forma sencilla por parte de los seres humanos.

Atendiendo a esta necesidad, se completa el módulo de depuración con dos objetos estáticos que permiten el formateo de fechas y números enteros en un formato fácilmente legible por los seres humanos. Para ello, hacemos uso de las clases `SimpleDateFormat` y `DecimalFormat` respectivamente.

```
1  /**
2   * Variable de formato para número flotantes
3   */
4  public static DecimalFormat df = new DecimalFormat("#.#####");
5  /**
6   * Variable de formato para fechas
7   */
8  public static SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MMdd HH:
    mm:ss");
```

Figura 7.7

Código: Sistema de depuración: formateo de fechas y decimales

7.3.1 Ejemplo de uso

A continuación se muestra un ejemplo de uso de este mecanismo para el formateo de fechas y decimales un formato amigable con el ser humano.

```
1 // ----- //
2 String label = Debug.sdf.format(startDate);
3
4 String valor = Debug.df.format(valorFlotante);
5
6 // ----- //
```

Figura 7.8

Código: Sistema de depuración: formateo de fechas y decimales

Parte III

ALMACENAMIENTO Y OPTIMIZACIÓN DEL ACCESO A DATOS

8

BASE DE DATOS LOCAL: MYSQL

ÍNDICE DE CAPÍTULO

8.1	Esquema de base de datos	82
8.1.1	Nodo	82
8.1.2	Dispositivos	83
8.1.3	Pasos	83
8.2	Optimizaciones en la base de datos	84
8.2.1	Optimizado de consultas con ventanas temporales	85
8.2.2	Optimización de consultas de recurrencias	88
8.2.3	Particionado de datos	90
8.2.4	Reducción del tamaño de las bases de datos	93
8.3	Procedimientos de procesamiento de datos	95
8.3.1	Procedimiento: Agrupación Pasos Por intervalos	95
8.3.2	Procedimiento: Cálculo de trazas para pasos entre nodos	98

Cómo se ha comentado anteriormente, para almacenar la base de datos local se emplea una instalación de MySQL donde se ha quedado con mimo la configuración del entorno para hacer que el gestor de base de datos se comporte de la manera más eficiente posible en el entorno que nos encontramos.

En este capítulo mostraremos las optimizaciones y configuraciones que se han realizado en el gestor de base de datos, así como la estructura y procedimientos empleados para el procesamiento de los datos.

8.1 ESQUEMA DE BASE DE DATOS

En esta sección mostraremos el esquema de base de datos empleado en cada tabla.

8.1.1 Nodo

Esta tabla es la encargada de almacenar la información de los nodos del sistema. Consta de los siguientes campos, cuya relación puede observarse en la Figura 8.1:

idnodo (BIGINT 20) Identificador del nodo, es unico para cada nodo-sensor.

latitud (FLOAT) Posición geográfica del nodo: latitud.

longitud (FLOAT) Posición geográfica del nodo: longitud.

nombre (CHAR 255) Nombre del nodo, en formato amigable con el ser humano. Suele hacer referencia a la ubicación del nodo así como al tipo de sensor que emplea para la detección.

Figura 8.1
Esquema de la
base de datos
local: Nodos

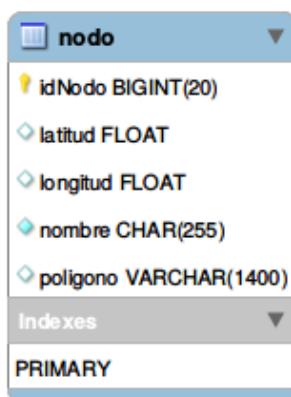




Figura 8.2
Esquema de la base de datos local: Dispositivos

poligono (VARCHAR 14000) Representación del nodo en el mapa usando notación KML.

8.1.2 Dispositivos

Es la tabla encargada de capturar la información relativa a cada dispositivo capturado por los nodos. Consta de los siguientes campos, cuya relación puede observarse en la Figura 8.2:

iddispositivo (CHAR 255) Identificador del dispositivo, es único para cada dispositivo. Suele ser calculado mediante un cifrado de la clave de identificación del dispositivo (por ejemplo, la dirección MAC en sensores tipo Bluetooth o Wifi).

majordeviceclass (CHAR 10) Tipo de dispositivo.

minordeviceclass (CHAR 5) Tipo de dispositivo.

serviceclass (CHAR 10) Clase de servicio.

fabricante (CHAR 250) Nombre del fabricante del dispositivo.

La naturaleza de los campos **majordeviceclass**, **minordeviceclass** y **serviceclass** se describe en el Anexo A, lo único que es necesario conocer es que son empleados por sensores bluetooth para conocer la naturaleza del dispositivo capturado.

8.1.3 Pasos

Es la tabla encargada de capturar la captación de un determinado dispositivo en un nodo concreto en un instante de tiempo determinado. Consta de los siguientes campos, cuya relación puede observarse en la Figura 8.3:

idnodo (BIGINT 20) Identificador del nodo, es único para cada nodo-sensor.

Figura 8.3
Esquema de la
base de datos
local: Dispositivos



iddispositivo (CHAR 50) Identificador del dispositivo, es único para cada dispositivo. Suele ser calculado mediante un cifrado de la clave de identificación del dispositivo (por ejemplo, la dirección MAC en sensores tipo Bluetooth o WiFi).

tfín (BIGINT 20) Instante de tiempo en el que se perdió la señal del dispositivo capturado. Expresado en formato unixtimestamp con precisión de milisegundos

tinicio (BIGINT 20) Instante de tiempo en el que se detectó por primera vez la señal del dispositivo capturado. Expresado en formato unixtimestamp con precisión de milisegundos

Cómo se puede observar, no se dispone de una única clave primaria para esta tabla, sino que cada tupla es representada de forma inequívoca por la convicción de los identificadores de Dispositivo y Nodo, y por el instante de tiempo. Es decir, no puede aparecer dos veces detectado el mismo dispositivo en el mismo instante de tiempo en el mismo nodo.

8.2 OPTIMIZACIONES EN LA BASE DE DATOS

Aunque el esquema presentado es bastante simple, se ha realizado un estudio con el fin de optimizar al máximo las tablas y las consultas relacionadas con la lectura de los datos, que como se ha comentado anteriormente, es una pieza crítica en nuestro sistema.

Es por ello que en este apartado se presentarán las optimizaciones llevadas a cabo en la estructura de la base de datos. Todas ellas mantienen en común que hacen uso de características avanzadas del gestor de Base de Datos, que no suelen ser habilitadas ni configuradas por defecto, pero que suponen una mejoría significativa en eficiencia.

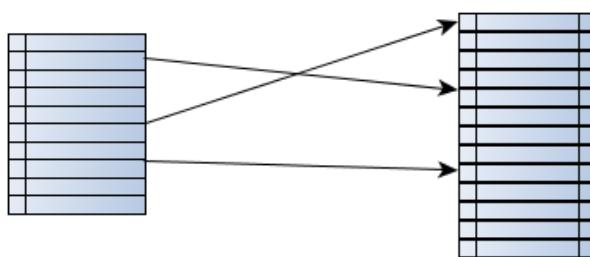


Figura 8.4
Funcionamiento de un índice Hash en la localización de tuplas

8.2.1 Optimizado de consultas con ventanas temporales

El marco temporal se encuentra presente en la mayoría de las peticiones que se realizan a la base de datos, esto es, las peticiones suelen requerir los datos de un intervalo concreto de tiempo. De esta forma, en la gran mayoría de peticiones del sistema, estamos interesados en un periodo de tiempo concreto, como por ejemplo los datos de un día en concreto, o de una hora o de una semana.

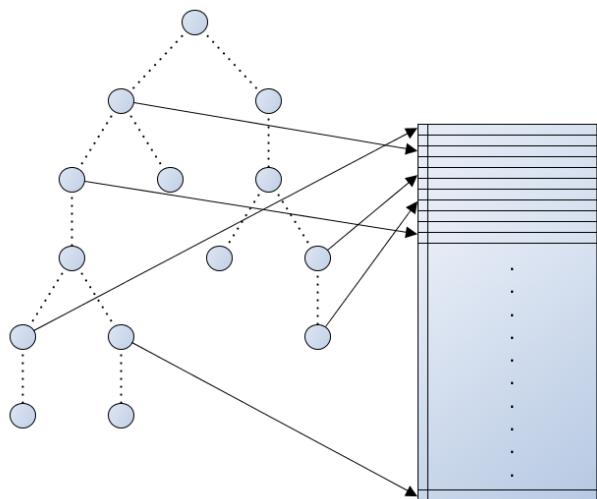
Un mecanismo muy usado en las bases de datos para optimizar las consultas, es configurar un índice en aquellos parámetros que determinan los campos de las búsquedas, es decir las condiciones WHERE. Sin embargo, el índice común para MySQL opera según una tabla HASH. Un índice basado en HASH determina mediante una función matemática la posición de una tupla determinada en función de la clave o KEY. Los índices basados en HASH permiten accesos a tuplas individuales y únicas en $O(1)$ (en orden constante), lo que supone un tiempo despreciable. Este tipo de índice priorizan los operadores de igualdad (o diferencia).

Sin embargo, para devolver listados de tuplas necesitan explorar todas las tuplas existentes por lo que el orden de eficiencia es $O(n)$. En la figura 8.4, se puede observar este mecanismo.

Sin embargo, este mecanismo no es eficiente para la localización de intervalos de tuplas. En su lugar se propone una implementación de índice basada en un árbol binario balanceado (en adelante B-TREE). Esta estructura es soportada de forma nativa por el gestor de base de datos, pero habitualmente no suele ser empleada a no ser que se indique expresamente, debido a que no es muy eficiente en la localización de tuplas individuales.

Esto es debido a que un índice basado en un B-TREE resuelve la posición de una tupla recorriendo un árbol binario balanceado, ordenado respecto al valor la clave o que define el índice. Esta organización permite un acceso a una tupla concreta en $O(\log n)$.

Figura 8.5
Funcionamiento de un índice basado en BTREE en la localización de tuplas



Como se puede observar, supone un orden de eficiencia peor que el índice basado en tabla HASH, y es por ello que no se configura por defecto. Sin embargo, un B-TREE permite recuperar intervalos de datos en $O(2\log n)$, lo cual es mucho más eficiente que el índice basado en HASH.

Por tanto, para recuperar las tuplas comprendidas entre un marco de valores, se tienen que localizar las tuplas correspondientes a dichos valores. Una vez localizadas las tuplas en el árbol, todas las tuplas comprendidas realizando un recorrido en enorden en el árbol desde un valor al otro, son las tuplas que corresponden con esos valores deseados. Además, las tuplas serán recorridas ordenadas según el orden del índice.

En la figura 8.5 se observa el comportamiento de un índice basado en B-TREE.

Experimento para ver la eficiencia

Debido a la dificultad para hacer experimentos en el entorno en caliente, se realizó una copia de 100000 de tuplas de la tabla *paso* con la que poder experimentar. Se realizan dos tablas exactamente iguales, con la única salvedad de que una tiene implementada un índice basado en BTREE y la otra implementa un índice por defecto.

Se ejecuta en ellas la siguiente consulta SQL:

```
1 mysql> select * from paso where tinicio BETWEEN '1352970861377' AND '  
1353020861377';
```

Figura 8.6

Consulta a ejecutar en la experimentación de índices. Dicha consulta solicita los dispositivos que han pasado entre dos fechas concretas, expresadas en formato UNIX_TIMESTAMP con precisión de milisegundos

En la siguiente tabla puede observar la diferencia de tiempo entre ambas consultas:

Tiempo (segundos)	
Índice por defecto	Índice basado en B-TREE
1.96	0.50

Tabla 8.1

Tiempos en la comparativa entre índice por defecto e índice basado en B-TREE

Se hace necesario recordar que en nuestro gestor de base de datos la caché de consultas está deshabilitada, por lo que en ningún momento influye en la petición. Además, de que son tablas independientes.

Sin embargo, al estar solicitando las tuplas, se está añadiendo tiempo de “transito”, por lo que hacemos el siguiente experimento con el fin de minimizar el tiempo dedicado a tareas que no sean la localización de los nodos. Se ejecuta por tanto en ellas la siguiente consulta SQL:

```
1 mysql> select count(*) from paso where tinicio BETWEEN '1352970861377' AND '  
1353020861377';
```

Figura 8.7

Consulta a ejecutar en la experimentación de índices II. Dicha consulta cuenta el número dispositivos que han pasado entre dos fechas concretas, expresadas en formato UNIX_TIMESTAMP con precisión de milisegundos

En este caso, al no estar devolviendo las tuplas encontradas, sólo contándolas el tiempo de transmisión de la petición se reduce al mínimo (un valor entero). Ejecutamos la sentencia en las dos tablas de las que disponemos.

Tiempo (segundos)	
Índice por defecto	Índice basado en B-TREE
1.93	0.01

Tabla 8.2

Tiempos en la comparativa entre índice por defecto e índice basado en B-TREE
II

Cómo se puede observar, el tiempo necesario para localizar nos nodos entre dos intervalos es irrisorio comparando el B-TREE con un índice por defecto. Notar, que el tiempo entre los dos experimentos es tan parecido en el caso del índice HASH, es debido a que MySQL procesa las transmisiones según va recuperando datos para transmitir. Es decir, una vez localizado una tupla empieza a transmitirla.

Se puede ver por tanto, que el empleo de un índice basado en B-TREE para el tratamiento de la ventana temporal es una herramienta muy eficiente. Y estamos hablando de un conjunto de datos relativamente pequeño frente a las búsquedas en millones de tuplas que esperamos realizar en el sistema.

Un problema que suelen acarrear los índices basado en B-TREE, es que las inserciones en nodos intermedios son bastante menos eficientes, debido a que se hace necesario rebalancear el árbol binario. En nuestro sistema, por fortuna, rara vez se producen inserciones en nodos intermedios ya que los valores insertados suelen pertenecer a instantes de tiempo más modernos que los almacenados en la base de datos y la inserción en la última rama supone un rebalanceo mucho más eficiente. Por tanto, nuestro sistema no se ve gravemente perjudicado por el empleo de este índice.

A día de hoy, con más 74 millones de tuplas en la base de datos, el tiempo de búsqueda en dicho intervalo sigue siendo inferior a los 0.01 segundos.

8.2.2 Optimización de consultas de recurrencias

Además del uso de las ventanas temporales en la mayoría de las consultas realizadas a nuestra base de datos, parte de nuestro cómputo residía en la búsqueda de recurrencias de un dispositivo capturado en distintos nodos. Supongamos así que un vehículo pasa por los nodos A y B en un corto periodo de tiempo y nos interesa conocer el tiempo que ha tardado en ir desde un

nodo al otro. A esta ruta de un dispositivo entre dos nodos, lo denominamos trazas.

Las trazas resultan muy interesantes de estudiar, pues nos permiten obtener estadísticos interesantes sobre el flujo de movimiento de los vehículos dentro del mapa donde se hayan ubicados los nodos. Sin embargo, computacionalmente raelizar dicha búsqueda en nuestro conjunto de datos supone un orden de eficiencia $O(n^2)$ pues para cada tupla se debe buscar entre las otras tuplas si el `idDispositivo` es el mismo. Más adelante, en la Sección 8.3 se abordará un procedimiento SQL que optimiza este tipo de consultas para la búsqueda de todas las trazas.

Sin embargo, la localización de una sola traza (su existencia o inexistencia) en un entorno no optimizado resulta inviable. Ejecutemos la siguiente consulta SQL para localizar las dos primeras trazas existentes en todo nuestro conjunto de datos de pruebas, que recordemos es de 100000 registros.

```
1 mysql> SELECT t1.idDispositivo, t1.idNodo as Origen,t1.tinicio as
   2   Origen_inicio, t1.tfin as Origen_fin, t1.tfin-t1.tinicio as Origen_dif,
   3   t2.idNodo as Destino, t2.tinicio as Destino_Inicio, t2.tfin as
   4   Destino_fin, t2.tfin-t2.tinicio as Destino_dif, from_unixtime(t1.
   5   tinicio/1000) as Origen_fecha, from_unixtime(t2.tinicio/1000) as
   6   Destino_fecha, t2.tinicio - t1.tinicio as Diferencia FROM paso as t1
   7   INNER JOIN paso as t2 ON t1.idDispositivo = t2.idDispositivo and t1.
   8   idNodo <> t2.idNodo and t2.tinicio - t1.tinicio BETWEEN 0 AND 3600000
   9   LIMIT 2;
```

Figura 8.8

Consulta a ejecutar en la experimentación de índices III. Dicha consulta busca en la base de datos dispositivos que hayan sido detectados por dos nodos distintos en una diferencia de tiempo de no más de una hora, devolviendo las 2 primeras ocurrencias.

Observar que en la consulta hacemos uso del mecanismo INNER JOIN para cotejar la tabla sobre si misma. Mecanismos de consulta como este nos hicieron decantarnos por un sistema SQL frente a las alternativas NoSQL disponibles en el mercado.

Se ejecuta la consulta superior, obteniéndose un tiempo de 58.69s que resulta demasiado elevado como para poder acercar al tiempo real este tipo de procesamiento.

Se realiza por tanto un estudio, para buscar la manera de optimizar la ejecución de dichas consultas. Es necesario notar que toda optimización realizada mediante la creación de índices tiene un consumo de espacio en memoria para alojar y mantener dichos índices. Es por ello que en la Sección 8.2.4 se realiza un estudio del espacio en disco por tupla, con el fin de intentar minimizar el espacio ocupado de forma inútil.

Se decide por tanto realizar un índice para el `idDispositivo` de forma que la recuperación de las tuplas con igual `idDispositivo` sea mucho más eficiente. De esta forma se le proporciona al gestor de base de datos de una estructura donde para cada `idDispositivo` puede localizar las tuplas que hacen referencia a este dispositivo en concreto.

Ejecutamos nuevamente la consulta reflejada en la Figura 8.14 habiendo realizado la optimización mediante un índice para el `idDispositivo`, consiguiendo un tiempo de 0.02s. Con la configuración de este mecanismo, se ha conseguido casi una ganancia de 3000 magnitudes en la eficiencia de la ejecución de este tipo de consultas, acercando el cómputo a prácticamente un tiempo constante, lo que nos permite realizar este cómputo prácticamente en tiempo real.

8.2.3 Particionado de datos

Durante el diseño del esquema de la base de datos se intentó simplificar el modelo lo máximo posible, minimizando el número de tablas en lo posible. Es por ello que se decidió que la tabla `paso` sería el gran contenedor que almacenase toda la información sobre el paso de dispositivos. Sin embargo, tal decisión nos imponía ciertas limitaciones debido a las características de las tablas. MySQL por defecto impone una limitación en el tamaño de archivo de 4GB como tamaño máximo. Esto limita el tamaño de una tabla InnoDB a 4GB como máximo de datos.

Para evitar esta limitación, MySQL dispone de un procedimiento de particionado, que permite particionar una tabla hasta en 1024 ficheros físicos acción que es necesario indicarle al gestor de base de datos, pues no la aborda de por defecto. Además, desde la versión 5.1 de MySQL es posible indicarle al gestor de base de datos como se desea que se realice el particionado de datos, pudiendo elaborar mecanismos muy eficientes si se aprovecha el sistema de particionado para emplear el principio de localidad espacial en disco en las consultas más frecuentes.

En la Figura 8.9 se observa como los datos son almacenados en disco según van llegando las peticiones de inserción. Recorremos que el proceso de sincronización mostrado en el Capítulo 5 presenta un proceso en el cual se realizan peticiones para una ventana temporal para cada nodo de forma secuencia. Es por ello, que los identificadores de nodos se presentan de forma muy parecida a la que se aprecia en la Figura 8.9, donde cada número indica el nodo que ha capturado el dispositivo.

.....0987
.....0987
....0123
....0123
....0987
....0123
....0476
....0987
....0476
....0476
....0123
....0476
....0987
....0123
.....
.....
.....
.....
.....
.....
.....
.....
.....

Figura 8.9
Almacenamiento
de una tabla sin
realizar particiona-
do

En la figura se ha potenciado, pero se hace evidente observar que se realizarán rachas en las cuales las tuplas consecutivas pertenezcan al mismo nodo (pues han sido introducidas una tras otras). Este comportamiento, hace que para leer los datos de una ventana temporal mayor que la ventana de inserción, sea necesario leer en el disco varias zonas no secuenciales.

Usando el mecanismo de particionado ofrecido por MySQL, hacemos corresponder un nodo a cada partición de datos, de forma como se observa en la Figura 8.10. Esto nos permite que todas las tuplas (pasos) correspondiente al mismo nodo se almacenen en el mismo archivo físico, y por tanto, es esperable que se encuentren en sectores consecutivos del disco duro. Además, aumentamos por 1024 el tamaño máximo de almacenamiento como se muestra en la sección 8.2.4.

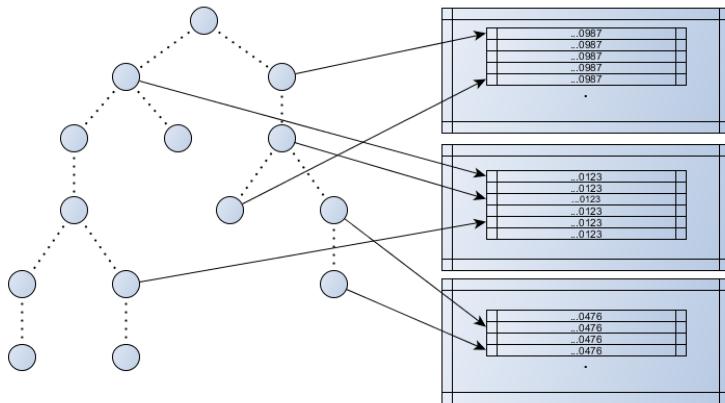
Pero además, supone otra ventaja adicional. Debido a nivel físico son archivos distintos, MySQL es capaz de trabajar con bloqueos a nivel de partición. De esta manera, para el gestor de base de datos es posible bloquear una partición para escritura exclusiva, conservando accesible el resto de particiones.

Además, en consultas como la presentada en la Sección 8.2.2 se ven muy beneficiadas debido al sistema RAID1 con el que se ha dotado al servidor como se explica en la Sección 4.1.1 pues el gestor puede leer cada partición de un disco distinto, ya que a nivel físico son archivos distintos.

Por último, este es un mecanismo que se está potenciando y mejorando en las versiones de pruebas de MySQL debido a las ventajas que supone.

En nuestro sistema, hemos realizado un particionamiento de 64 unidades físicas, quedando aún posibilidad de emplear el particionamiento o bien para incorporar mayor cantidad de nodos, o bien para hacer uso de esta potente técnica de particionamiento en otros aspectos de los datos.

Figura 8.10
Funcionamiento de una tabla particionada en función del idNodo.



8.2.4 Reducción del tamaño de las bases de datos

Normalmente el tamaño de que ocupa una base de datos no suele preocupar a los diseñadores de la misma. Sin embargo, dada la naturaleza del proyecto ha sido un aspecto que también ha sido cuidado de forma meticulosa. Se presenta a continuación los tamaños asociados a la tabla pasos presentada al comienzo de este capítulo:

Tuplas	Tamaño (Bytes)			
	Tupla	Total	Índice	Libre
9632529	101	980418560	0	2721054720

Tuplas	Tamaño (MBytes)			
	Tupla	Total	Índice	Libre
9632529	101	935MB	0	2595 MB

Tabla 8.3
Tamaños asociados a la tabla paso sin optimizaciones

A continuación se presenta la tabla pasos una vez realizadas las optimizaciones presentadas a lo largo de este capítulo, es decir, los índices para marcos temporales y dispositivos y el mecanismo de particionado.

Tuplas	Tamaño (Bytes)			
	Tupla	Total	Índice	Libre
9632529	101	980418560	1824426050	160353640529

Tuplas	Tamaño (MBytes)			
	Tupla	Total	Índice	Libre
9632529	101	935MB	1739 MB	152925 MB

Tabla 8.4
Tamaños asociados a la tabla paso con las optimizaciones presentadas anteriormente, es decir, los índices para ventanas temporales y para la gestión eficiente de reincidencia de dispositivos, así como el sistema de particionado.

Para empezar observemos que se ha mejorado la capacidad máxima de nuestra tabla de poco más de 2.5GB a casi de 150GB de almacenamiento para la tabla pasos. Recordemos que sólo hemos empleado un particionamiento de 64 bloques físicos. Si realizásemos un particionamiento de lo máximo permitido actualmente por MySQL, tendríamos una capacidad total de 2.33TB.

Sin embargo, como se anunció anteriormente, los índices ocupan memoria para ser almacenados, por lo que se ha pasado de no gastar memoria en los índices a estar almacenando 1739MB para índices por cada 935MB de datos. Se observa que el tamaño de datos por cada tupla es de 101bytes para los datos y unos aproximados 189bytes para la estructura de índices.

Se realiza un estudio con el fin de intentar reducir el tamaño de los índices y de la tupla. Se realiza una optimización en los tipos de datos empleados en las tuplas, lo cual permite reducir de 101bytes a 93bytes de tamaño cada tupla.

Tuplas	Tamaño (Bytes)			
	Tupla	Total	Índice	Libre
9632529	93	896614400	1679917056	174147502080

Tuplas	Tamaño (MBytes)			
	Tupla	Total	Índice	Libre
9632529	93	855 MB	1602 MB	166080 MB

Tabla 8.5

Tamaños asociados a la tabla *paso* con las optimizaciones presentadas anteriormente y realizando una optimización para reducir el tamaño por tupla.

Para entender como se logra reducir el tamaño del índice, es necesario entender como funciona un índice a nivel interno. Hemos comentado que cada índice permite referenciar de forma eficiente un tupla concreta, ya sea mediante una función hash o realizando un recorrido en un árbol binario. Sin embargo, hemos obviado cómo se referencia una tupla de forma única. Para ello se hace uso del índice primario (**PRIMARY KEY**).

Visualizando la Figura 8.3 podemos observar que el índice primario de nuestra tabla *paso* implica la utilización del *idNodo*, *idDispositivo* y el *tinicio*. Esto supone que en la entrada de cada índice, tenemos que almacenar los bytes correspondientes a estos tres campos, pues son los que permiten identificar una tupla. Dado los tipos de datos empleados, consideramos que estamos gastando unos 58bytes para referenciar cada tupla.

Se plantea por tanto la creación de un identificador adicional que permita identificar cada tupla. Se elige un tipo de dato **UNSIGNED BIGINT** debido a que es el tipo de datos que permite referenciar el mayor cantidad de tuplas y que supone un coste de sólo 8bytes.

Tuplas	Tamaño (Bytes)			
	Tupla	Total	Índice	Libre
9632529	99	957251584	897024000	174147502080
Tuplas	Tamaño (MBytes)			
	Tupla	Total	Índice	Libre
9632529	99	913 MB	855 MB	166080 MB

Tabla 8.6

Tamaños asociados a la tabla paso con las optimizaciones presentadas anteriormente, realizando una optimización para reducir el tamaño por tupla e implementando un nuevo índice

Realizando esta optimización se reduce el tamaño del índice a 855MB frente a los 1602MB que ocupaba anteriormente. Lo cual supone un ahorro significativo de memoria.

8.3 PROCEDIMIENTOS DE PROCESAMIENTO DE DATOS

Se presentan a continuación dos procedimientos para el procesamiento de datos: para obtener resúmenes de los datos respecto a intervalos de tiempo y el método que calcula las trazas.

8.3.1 *Procedimiento: Agrupación Pasos Por intervalos*

Este procedimiento permite calcular de forma eficiente los dispositivos que han sido localizados en un intervalo de tiempo, agrupados según una ventana temporal. Es decir, permite por ejemplo calcular los vehículos que han pasado en la última semana, agrupándolos en intervalos de 1 hora, o intervalos de 1 día.

Su funcionamiento eficiente, permite el tratamiento de los datos de forma rápida para la utilización en métodos más complicados, como los de predicción o los de subida a la nube.

Se presentan tres alternativas, en función de si se desean los intervalos de un solo nodo, de todos los nodos, o independientemente de los nodos.

Un solo nodo

Este procedimiento permite, para un nodo indicado, obtener un resumen del tránsito de dispositivos agrupamos por un intervalo de tiempo especificado.

```
1 CREATE DEFINER='root'@'localhost' PROCEDURE 'agrupaPasosPorIntervalosNodo'(
2     in fechaMIN Varchar(40), in fechaMAX Varchar(40), in intervalo INT, in
3     nodo BIGINT(20) )
4 BEGIN
5     DECLARE inter INT DEFAULT intervalo*1000*60;
6     SELECT FROM_UNIXTIME(round(tinicio/inter)*inter/1000) as Intervalo ,
7             count(*)
8     FROM paso
9     WHERE
10         idNodo = nodo
11         AND tinicio
12         BETWEEN UNIX_TIMESTAMP(fechaMIN)*1000
13             AND UNIX_TIMESTAMP(fechaMAX)*1000
14
15         GROUP BY
16             round(tinicio/inter)
17             #INTO OUTFILE '/tmp/pasos.csv'
18             #FIELDS TERMINATED BY ','
19             #ENCLOSED BY ""
20             #LINES TERMINATED BY '\n'
21 ;
22 END
```

Figura 8.11
Procedimiento para el agrupamiento de los pasos por intervalos de tiempo, para un solo nodo.

Todos los nodos

Este procedimiento permite, para todos los nodos, obtener un resumen del tránsito de dispositivos agrupamos por un intervalo de tiempo especificado. Este procedimiento no distingue entre nodos, es decir, no indica cuantos dispositivos han pasado por cada nodo en el intervalo determinado.

```

1 CREATE DEFINER='root'@'localhost' PROCEDURE 'agrupaPasosPorIntervalos'(in
2   fechaMIN Varchar(40), in fechaMAX Varchar(40), in intervalo INT )
3 BEGIN
4   DECLARE inter INT DEFAULT intervalo*1000*60;
5   SELECT FROM_UNIXTIME(round(tinicio/inter)*inter/1000) as Intervalo ,
6         count(*)
7   FROM paso
8     WHERE tinicio
9       BETWEEN UNIX_TIMESTAMP(fechaMIN)*1000
10      AND UNIX_TIMESTAMP(fechaMAX)*1000
11 GROUP BY
12   round(tinicio/inter);
13 END

```

Figura 8.12

Procedimiento para el agrupamiento de los pasos por intervalos de tiempo, para todos los nodos sin distinción entre ellos.

Nodos separados

Este procedimiento permite, para todos los nodos, obtener un resumen del tránsito de dispositivos agrupamos por un intervalo de tiempo especificado, distinguiendo la cantidad de dispositivos que ha detectado cada nodo concreto.

```

1 CREATE DEFINER='root'@'localhost' PROCEDURE '
2   agrupaPasosPorIntervalosNodosSeparados'(in fechaMIN Varchar(40), in
3   fechaMAX Varchar(40), in intervalo INT )
4 BEGIN
5   DECLARE inter INT DEFAULT intervalo*1000*60;
6   DECLARE corte INT DEFAULT 19;
7
8   IF intervalo >= 1440 THEN SET corte = 10; END IF;
9
10  SELECT SUBSTR(FROM_UNIXTIME(truncate(tinicio/inter,0)*inter/1000),1,corte)
11    as Fecha, __paso.idNodo ,  count(*) as Total, latitud,longitud,nombre,
12    poligono
13   FROM __paso, nodo
14     WHERE
15       tinicio
16         BETWEEN UNIX_TIMESTAMP(fechaMIN)*1000
17            AND UNIX_TIMESTAMP(fechaMAX)*1000
18         AND
19           __paso.idNodo = nodo.idNodo
20 GROUP BY
21   idNodo,
22   truncate(tinicio/inter,0)
23 ORDER BY Fecha ASC;
24
25 END

```

Figura 8.13

Procedimiento para el agrupamiento de los pasos por intervalos de tiempo, para todos los nodos con distinción entre ellos.

8.3.2 Procedimiento: Cálculo de trazas para pasos entre nodos

Este procedimiento permite el cálculo de las trazas que han ocurrido entre los diversos nodos del sistema en una ventana temporal, con un límite de tiempo determinado. Este límite indica cuando tiempo puede pasar como máximo en la detección en dos nodos distintos para ser considerado una traza.

Este procedimiento hace uso de un mecanismo de tablas temporales basadas en memoria que es bastante eficiente, debido a que no requiere volcar los datos que está calculando en ningún archivo físico.

El uso de este procedimiento es muy eficiente, al basarse en el empleo de la técnica MapReduce para dividir los conjuntos de datos en partes más pequeñas, procesarlas e integrarlas posteriormente en el conjunto de datos finales. Lo que habitualmente se llama reduce y vencerás. Para ello, lo que hace es realizar cortes del tamaño de la ventana temporal más el tiempo que determina la existencia o no de traza y va procesando esas ventanas una a una, en lugar de realizar el JOIN sobre todo el conjunto de datos.

A pesar de su eficiencia, la búsqueda de trazas sigue siendo costosa en tiempo de cómputo, debido a lo cual aún no se ofrece en tiempo real la información de las trazas.

```

1 CREATE DEFINER='root'@'localhost' PROCEDURE 'localizaTrazasNodos2'(in
2   fechaMIN Varchar(40), in fechaMAX Varchar(40), in intervalo INT)
3 BEGIN
4
5   DECLARE inter INT DEFAULT intervalo*1000*60;
6   DECLARE corte INT DEFAULT 19;
7   DECLARE fMax bigint(20) DEFAULT UNIX_TIMESTAMP(fechaMAX)*1000;
8   DECLARE fMin bigint(20) DEFAULT UNIX_TIMESTAMP(fechaMIN)*1000;
9   DECLARE i bigint(20) DEFAULT fMIN;
10
11  IF intervalo >= 1440 THEN SET corte = 10; END IF;
12
13  DROP TEMPORARY TABLE IF EXISTS trazas;
14  CREATE TEMPORARY TABLE trazas (Origen bigint(20), tiempo bigint(20),
15    Destino bigint(20), Destino_tiempo bigint(20)) ENGINE=MEMORY;
16
17  WHILE (i+inter<=fMaX) DO
18    INSERT INTO trazas
19      SELECT STRAIGHT_JOIN
20        t1.idNodo as Origen,
21        t1.tinicio as tiempo,
22        t2.idNodo as Destino,
23        t2.tinicio as Destino_tiempo
24      FROM
25        (SELECT idNodo,tinicio,idDispositivo FROM __paso
26         WHERE tinicio
27             BETWEEN i
28               AND i+inter
29     ) as t1
30     INNER JOIN
31       (SELECT idNodo,tinicio,idDispositivo FROM __paso
32         WHERE tinicio
33             BETWEEN i-inter
34               AND i+inter+inter
35     ) as t2
36     ON
37       t1.idDispositivo = t2.idDispositivo
38       AND t1.idNodo <> t2.idNodo
39       AND (t2.tinicio - t1.tinicio) BETWEEN 0 AND inter;
40
41     SET i = i + inter;
42  END WHILE;
43
44  SELECT t.Fecha, t.Origen, t.Destino, t.total, t.Diferencia, t1.latitud, t1.
45    longitud, t2.latitud, t2.longitud FROM
46    (SELECT SUBSTR(FROM_UNIXTIME(truncate(tiempo/inter,0)*inter
47      /1000),1,corte) as Fecha,
48      Origen,
49      Destino,
50      count(*) as total,
51      avg(Destino_tiempo - tiempo)/1000 as Diferencia
52    FROM trazas
53    GROUP BY Origen, Destino, truncate(tiempo/inter,0)
54    ORDER BY fecha DESC) as t
55    INNER JOIN (SELECT latitud,longitud,idNodo from nodo) as t1 ON t.Origen = t1
56      .idNodo
57    INNER JOIN (SELECT latitud,longitud,idNodo from nodo) as t2 ON t.Destino =
58      t2.idNodo
59
60  ;
61 END

```

Figura 8.14
Procedimiento el cálculo de trazas entre pasos.

BASE DE DATOS EN LA NUBE: GOOGLE FUSION TABLES

ÍNDICE DE CAPÍTULO

9.1	Ventajas de uso de Google Fusion Tables	102
9.1.1	Defición	102
9.1.2	Beneficios de uso	103
9.2	Google Fusion Tables	105
9.2.1	Arquitectura y almacenamiento de datos	105
9.2.2	Procesado de consultas	109
9.2.3	Transacciones	109
9.2.4	Visualización de los datos	109
9.2.5	Características geolocalizadas	110
9.2.6	Integración con aplicaciones de terceros	110
9.3	Uso de Fusion Tables en el Proyecto	110

Si bien usamos una versión muy optimizada de MySQL para el almacenamiento y computo local en lugar de alternativas basadas en modelos de base de datos NoSQL muchos más empleadas en entornos de BigData, no podemos negar que su utilidad y eficiencia para algunas situaciones es muy superior a las ofrecidas por un sistema SQL.

Es por ello, que se decide emplear un gestor de base de datos distinto a MySQL para la publicación de los datos públicos.

De esta manera, el cómputo local se realiza en la base de datos local MySQL muy optimizada para el cómputo, y los datos procesados son almacenados en un sistema NoSQL alojado en la Nube.

Existen una gran cantidad de alternativas noSQL, sin embargo nos decantamos por una solución muy poco conocida de Google: Fusion Tables [20] [21].

9.1 VENTAJAS DE USO DE GOOGLE FUSION TABLES

9.1.1 Defición

Google Fusion Tables es un servicio basado en CLOUD para la administración e integración de datos tabulares potencialmente geolocalizados, el cual provee de diferentes maneras de visualizar, filtrar y procesar los datos almacenados. Soporta integración de datos de múltiples fuentes realizando “JOINS” cruzando tablas que pueden pertenecer a usuarios distintos. Este JOIN constituye una vista de datos cruzados, no un mecanismo de selección en consultas.

Los usuarios pueden mantener sus datos privados, compartidos con un conjunto de colaboradores o hacer los datos públicos accesibles para cualquier motor de búsqueda.

Google Fusion Tables ofrece también una API REST para administración de las tablas, ventanas de información de las plantillas y estilos de visualización. Ofrece también un sistema de consultas para gestionar las tuplas (inserción, actualización o eliminación) así como realizar selecciones de tuplas basadas en condiciones de datos o geográficos. Dicha consulta puede ser devuelta en CSV, JSON o ser usada de forma nativa por otros entornos de Google como *Google Maps* o *Google Chat Tools*. Esta funcionalidad permite una rápida y transparente representación y publicación de los datos.

9.1.2 *Beneficios de uso*

En esa sección se recoge a modo de resumen algunas de las características y ventajas que justifican el uso de esta plataforma.

Uso de datos públicos

Fusion Tables permite la búsqueda entre miles de tablas alojadas en Fusion Tables, o entre millones de tablas existentes en la web que se pueden importar en Fusion Tables. De igual manera, si se desea, se puede hacer que todos los datos alojados en Fusion Tables sean accesibles de forma pública.

Importar tus propios datos

Ya sea mediante la subida de un fichero CSV, una hoja de cálculo o un fichero KML, o bien mediante el uso de la API REST que permite insertar, actualizar, eliminar y realizar consultas de forma programada, con Fusion Tables se puede subir los datos de forma rápida y sencilla a la nube.

Visualizar de forma instantánea

De forma inmediata, una vez los datos han sido importados, se pueden visualizar los datos en gráficos o establecer una geolocalización en un mapa. Además, se pueden realizar filtros para visualizaciones más selectivas.

Publicación inmediata

Cualquier gráfico o mapa realizado puede ser embebido vía WEB o enviado vía correo electrónico, mostrando siempre los valores actualizados de la base de datos, sin tener que realizar ningún trámite adicional.

Cotejar datos con otras bases de datos

Si alguien tiene de datos diferentes sobre la misma entidad de datos o si se dispone los datos en distintas tablas, con Fusion Tables se puede cotejar los datos, haciendo un JOIN transparente entre ambas tablas, lo que generará una nueva vista.

Siempre actualizados

Cuando alguna de las bases de datos es actualizada (ya sea modificación o inserción de nuevos datos) las tablas cotejadas se actualizan de forma automática y transparente.

Compartir sólo los datos deseados

Si se necesita mantener alguno de los datos, se permite compartir sólo un conjunto de columnas de la tabla principal, que estará siempre actualizados respecto a la tabla original, pero con sus propios permisos de publicación.

Constante autoría de los datos

Fusion Tables permite mantener de forma constante de donde provienen los datos y a quien pertenecen. Tanto durante la importación y la visualización se muestran la autoría de los datos. Incluso si las tablas son mezcladas, se seguirá manteniendo la autoría de los datos originales, y así será mostrado.

Geolocalización transparente de los datos

Ya sean datos expresados en puntos, líneas, polígonos, direcciones, nombre de lugares, países o cualquier registro susceptible de almacenar datos geográficos, es automáticamente interpretado y geolocalizado en un mapa.

Personalización de los mapas

Aplicar colores o iconos basados en los datos. Hacer mapas de intensidades basadas en zonas. Usar polígonos KML para el dibujado sobre el mapa. Mostrar miles de trazas al mismo tiempo.

Compartir el mapa

Los mapas generados con Fusion Tables pueden ser embebidos en cualquier WEB, enviados mediante correo electrónico, ser salvado como un fichero KML para ser visto en cualquier software de geolocalización, o incluso usar Fusion Tables para la generación dinámica de KML ofrecidos como enlaces para mantener siempre un enlace al mapa constantemente actualizado.

Alta accesibilidad web

Fusion tables permite almacenar los datos y ofrecer un portal donde los usuarios puedan visualizar los datos sin necesidad de descargarlos. Pueden explorar los mapas, gráficos, realizar cálculos sobre ellos o hacer búsquedas y filtros sobre los datos para su posterior descarga. O si se desea, se puede impedir la descarga de los datos, permitiendo sólo su visualización ONLINE.

Difusión de los datos actualizados y de forma segura

En lugar de mantener miles de copias de los datos en diversos discos duros locales, los datos en Fusion Tables están siempre actualizados y seguros, mostrando siempre los últimos datos en todas las gráficas y mapas que han sido generadas de forma automática.

Posibilidades de expansión mediante la API

Cuando los datos se almacenan en Fusion Tables, se ofrece de forma automática una API para que los desarrolladores puedan acceder a los datos mediante REST. Ofertando tanto una forma pública (basa en token) como un sistema autenticado basado en OAuth. Únicamente mediante el sistema autenticado se podrá alterar los datos (modificar, eliminar, insertar,..)

9.2 GOOGLE FUSION TABLES

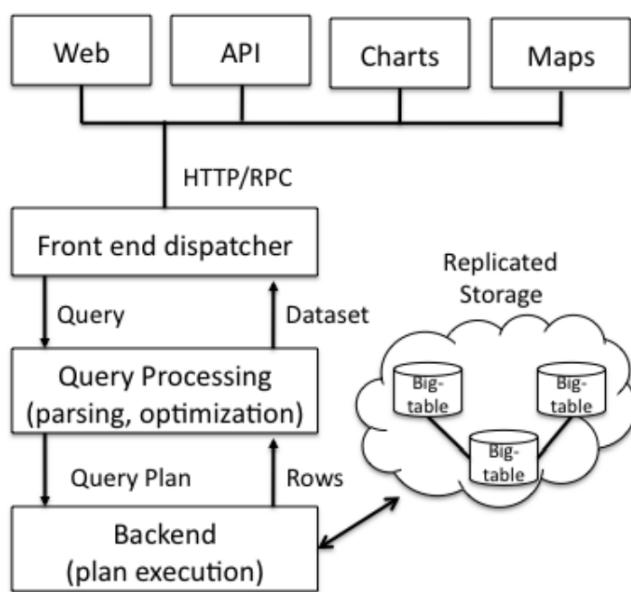
En esta sección se describirán aspectos más técnicos sobre Google Fusion Tables, con el fin de comprender el funcionamiento interno de la plataforma.

9.2.1 *Arquitectura y almacenamiento de datos*

Es necesario explicar (de forma breve) la arquitectura de Fusion Tables y el sistema de almacenamiento de los datos. Primero se describen los elementos de la pila de almacenamiento (Storage Stack) sobre los que se construye Fusion Tables: Bigtable y Megastore.

A continuación, se describe como se almacena en dicha estructura todas las tuplas de todas las tablas del usuario en una sola tabla Bigtable y todo el esquema en otra tabla Bigtable.

Figura 9.1
Arquitectura de almacenamiento de Google Fusion Tables



Arquitectura

En la Figura 9.1 se muestran los principales componentes arquitectónicos del servicio Fusion Tables. Las peticiones pueden ser originadas desde diversas fuentes:

- La vista web de Google Fusion Tables
- La API REST de comunicación
- Visualizaciones embebidas soportadas de forma nativa
 - Un mapa de Google Maps
 - Un gráfico de Google Chart

Todas son procesadas igualmente, con la salvedad de las peticiones sobre mapas, que se generan mediante consultas espacio/estructurales planificadas sobre las tablas del sistema. Es decir, se utilizan las coordenadas de visualización del mapa para la obtención de “imágenes” que serán geolocalizadas en el mapa con la información resultante.

En el resto de casos, el FrontEnd convierte las peticiones (query) de las distintas fuentes a una representación común, y lo envía al módulo de procesamiento de consultas (Query Processing Module) que genera una planificación para la consulta (Query Plan).

Dicho plan es ejecutado por el Back End haciendo uso de un conjunto sincrónico y replicado de servidores de almacenamiento Big

Table. El principal reto para la capa de almacenamiento, por tanto, es tratar con multitud de tablas, con diversidad de esquemas tamaños y naturaleza de las consultas.

Pila de almacenamiento

Fusion Tables se construye sobre dos capas distintas dentro de la infraestructura de almacenamiento de Google:

Big Table

Las tuplas almacenadas dentro de una estructura Big Table son de tipo {clave,valor}, las cuales se encuentran ordenadas en función de la clave y distribuidos por rangos (de la propia clave) en múltiples servidores. El valor, puede ser cualquier estructura compleja formada por valores atómicos permitidos.

El entorno BigTable provee de un única operación de escritura que inserta una tupla de forma atómica, esto es, en tiempo constante y sin posibilidad de interrupción.

Se provee también de tres mecanismos de lectura:

- Búsqueda por clave (Lookup by key) donde para una clave identificada se devuelve el valor asociado a dicha clave.
- Búsqueda por prefijo de clave (Lookup by key prefix) donde se devuelven los valores para los cuales el comienzo de su clave comienza con el prefijo buscado.
- Búsqueda por rango de clave Lookup by key range donde se devuelven los valores comprendidos entre las dos claves especificadas.

Además, el entorno BigTable almacena un sello de tiempo histórico (timestamp) para cada tupla. Así se podría entender una tupla como el conjunto {clave,valor,marcaDeTiempo}, siendo la marca de tiempo el instante en el que ha sido escrito. Esto es debido a que el sistema permite almacenar tuplas con claves coincidentes.

Megastore

Megastore es un conjunto de librerías situadas por encima de BigTable. En ellas se provee de primitivas de alto nivel, como índices secundarios (Sección 9.2.2), transiciones sobre múltiples tuplas (Sección 9.2.3) y replicaciones entre diversos servidores conservando la consistencia.

Almacenamiento de tuplas: Row Store

En Fusion Tables, todas las tablas de un usuario son almacenadas en una única tabla BigTable de nombre *Rows* (filas o tuplas). Cada tupla dentro de dicha tabla pertenece a una tupla en una tabla concreta del usuario. La clave empleada para la tupla, es la concatenación del identificador de la tabla y el identificador de la fila. Dicho identificador de fila es gestionada de forma interna y transparente para el usuario, que no necesita definir una “clave primaria” para su tabla.

Almacenamiento de esquemas: Schema Store

Para almacenar los esquemas de las tablas se hace uso de otra tabla BigTable, con una tupla por cada tabla almacenada en el sistema. El valor de la clave, es el identificador de la tabla.

En el valor, se almacenan las columnas y permisos para cada tabla. De cada columna se almacena el nombre y el tipo de dato preferente. En los permisos, se almacenan los usuarios asociados a sus diversos permisos sobre la tabla. Las tablas públicas constante de un registro especial indicando que son visibles por todo el mundo.

Con este sistema, se permite que el esquema de una tabla pueda evolucionar con el tiempo, pudiendo los usuarios añadir y eliminar columnas sobre sus datos.

Almacenamiento de vistas: View Store

Una de las principales funcionalidades de Fusion Tables es que permite que múltiples usuarios puedan cotejar sus datos “uniendo” sus tablas en una sola, incluso si dichas tablas pertenecen a usuarios distintos o incluso a usuarios terceros. Una tabla construida mediante uniones de tablas se denomina una vista, y dispone de un almacenamiento virtual. Esto es, las vistas no son almacenadas en la BigTable Rows de ninguno de los usuarios, solo se almacena su definición y permisos en la BigTable Schema. Las vistas tienen sus propios permisos, de igual manera que las tablas de usuarios.

Almacenamiento de comentarios: Comment Store

Para facilitar las colaboraciones, Fusion Tables permite los comentarios tanto de tablas, tuplas, columnas o celdas individua-

les. Todos los comentarios son almacenados en una tabla BigTable, donde la clave es el elemento comentado y el valor es el comentario.

9.2.2 *Procesado de consultas*

Fusion Tables se encuentra limitado a las primitivas de lectura que ofrece BigTable para dar soporte a un conjunto limitado de consultas SQL. Actualmente el sistema soporta selecciones (SELECT) sobre los valores de cualquier columna; así como agragaciones (GROUP BY) y uniones (JOINS) basados en la clave primaria.

La estrategia de ejecución de consultas se basa en traducir la consulta a procesar en un conjunto de las tres operaciones soportadas por las tablas BigTable presentadas anteriormente.

9.2.3 *Transacciones*

Fusion Tables está diseñado como una plataforma para el almacenamiento y gestión de datos con un fuerte carácter orientado a la colaboración y visualización de los datos de forma estructurada. Debido a ello que no se encuentra diseñado para ser un sistema de alto rendimiento en la transición y procesamiento de los datos. Esta naturaleza de Fusion Tables es la que nos hace mantener el doble sistema de almacenamiento presentado en este apartado.

El entorno MySQL anteriormente presentado y muy optimizado para el procesamiento y transición de datos; y el entorno Fusion Tables, más orientado a la visualización, representación y compartición de los datos.

9.2.4 *Visualización de los datos*

Como se ha repetido en numerosas ocasiones en este capítulo, uno de los principales potenciales de Fusion Tables es su capacidad de representar los datos inmediatamente se encuentran alojados en el servicio. El conjunto de visualizaciones permitidas para un conjunto de datos concreto es definido en función de los tipos de datos alojados en la tabla y los tipos requeridos para cada visualización.

Por ejemplo, para la visualización de un “nube de puntos.^{es} necesario que existan dos variables numéricas (una para eje de la gráfica). De manera similar, para poder representar los datos en un mapa, es necesario que la tupla se encuentre geolocalizada.

9.2.5 *Características geolocalizadas*

Uno de las grandes bazas de Fusion Tables, es su capacidad para trabajar con grandes conjuntos de datos geolocalizados. La forma de geolocalizar un dato, es mediante la inclusión en la tupla de una columna con información geográfica. Ya sea una mediante una dirección o unas coordenadas. Las coordenadas pueden estar asociadas a un punto, una línea o un polígono.

Fusion Table renderiza en el lado del servidor toda la información geográfica como una capa, dejando al cliente la tarea de representarla. De esta forma el grueso del cómputo es realizado por el servidor, resultando mucho más ligero para el cliente, lo cual facilita la implementación de clientes en dispositivos de sin grandes necesidades de cómputo, como dispositivos móviles.

9.2.6 *Integración con aplicaciones de terceros*

Para terminar, un aspecto de vital importancia de Fusion Tables es la capacidad de extender la funcionalidad básica de la plataforma gracias a su API. Esta API permite a desarrolladores externos de Google (como nosotros) escribir aplicaciones que hace uso de Fusion Tables como base de datos.

Esta API se describirá en detalle en el capítulo 13.

9.3 USO DE FUSION TABLES EN EL PROYECTO

Fusion Tables es elegido por tanto como entorno para la publicación de los datos en la Nube, lo que permite ofrecer la información calculada por el sistema visto en el Capítulo 5 en un entorno Web. Además, debido a su API de comunicación, es posible realizar un módulo encargado de la sincronización automática de la información en la nube a medida que se va calculando en el servidor local. Este sistema será descrito en detalle en el Capítulo 12.

Parte IV

PREDICCIÓN

10

PREDICCIÓN OFFLINE

ÍNDICE DE CAPÍTULO

- 10.1 Estudio preliminar series temporales **114**
 - 10.1.1 ARIMA **115**
 - 10.1.2 L-Co-R **116**
 - 10.1.3 Weka Forecasting **117**
 - 10.2 Predicción por días **117**
 - 10.3 Predicción por horas **121**
-

En los capítulos anteriores hemos descrito el sistema que monitorea el estado del tráfico mediante la adquisición y procesado del número de dispositivos que han sido detectados por los distintos sensores desplegados en el mapa. De esta forma ofrecemos un sistema que es capaz de recoger el estado del flujo del tráfico en instantes pasados de tiempo. Debido también a las optimizaciones del procesado de datos llevadas a cabo que lo acercan al tiempo real, es posible también hablar de un sistema capaz de mostrar el estado “actual” del tráfico.

Habiendo cubierto el “pasado” y el “presente”, en este capítulo abordaremos la predicción de los datos futuros en función de los datos pasados haciendo uso de *Técnicas de Predicción de Series Temporales*.

Sin embargo, es necesario entender que esta no ha sido la labor principal del estudiante en la presentación del sistema presentando, sino una aplicación práctica al sistema de procesado y almacenamiento de los datos.

Se presenta por tanto en este capítulo las predicciones “offline” realizadas sobre el sistema de datos desarrollado. Entendemos como predicción offline aquella que ha sido realizada con un conjunto de datos estático de forma manual. En el Capítulo (11) se presenta el módulo de predicción Online desarrollado para SiMa. Entendiendo como predicción Online, a la predicción realizada en tiempo real y de forma automática por el sistema en función de los datos recibidos.

10.1 ESTUDIO PRELIMINAR SERIES TEMPORALES

Debido a la gran cantidad de métodos y técnicas de predicción¹ existentes en la bibliografía, se hacía necesario realizar un estudio preliminar sobre un conjunto de datos reales obtenido con el sistema.

Principalmente estudiamos algoritmos de clustering (k-vecinos más cercanos, SOM [27], métodos de predicción de series temporales y aproximación funcional (ARIMA, Croston, Theta, Spline y L-Co-R [19] [18]), redes neuronales artificiales (MLP [14] [15], RBF [36]), y árboles de decisión (C4.5, [33] [34]).

Dado que los datos se recogen y almacenan manteniendo una marca de tiempo (esto es, pertenecen a una cronología), se apli-

¹ Con las que mi compañero en el proyecto J.Asensio tuvo que lidiar mucho más en profundidad.

caron métodos muy conocidos en la predicción de series temporales para hacer predicciones certeras de los flujos de tráfico.

La predicción de series temporales es un área de investigación en constante avance para conseguir desarrollar modelos eficaces en la predicción. Se distinguen dos tipos de técnicas de predicción: los métodos lineales y no lineales.

Entre todos esos métodos, ARIMA [12] es el método lineal más potente y con mayor éxito, además de uno de los métodos más trabajados y estudiados para la predicción.

Entre los métodos no-lineales se recogen los modelos autoregresivos [13, 16, 17, 40, 41]. Sin embargo los métodos no lineales se construyen sobre modelos muy complejos, haciendo uso de bases no robustas, que en ocasiones resultan muy difíciles de configurar y usar.

Concretamente se estudiaron los métodos ARIMA, Croston, Theta, Spline y L-Co-R [19] [18].

10.1.1 ARIMA

Arima es método de regresión lineal propuesto por Box y Jenkins [12] mediante un modelo autorregresivo integrado de media móvil o ARIMA (acrónimo del inglés autoregressive integrated moving average). Se trata de un modelo estadístico que utiliza variables y regresiones de datos estadísticos con el fin de encontrar patrones para una predicción hacia el futuro.

Este modelo trabaja en un sistema iterativo en tres estados consistente en:

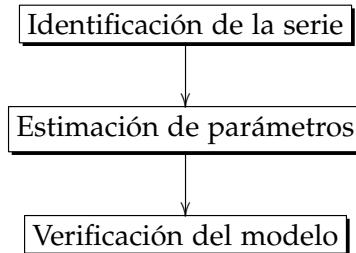


Figura 10.1
Tareas del método de predicción ARIMA

10.1.2 L-Co-R

L-Co-R (Lags COevolving with Rbfns) [28, 29] es un algoritmo que diseña RBFNs para la predicción de series temporales. Para ello obtiene el número apropiado de RBFs, un radio y centro para cada RBF, los pesos asociados a toda la red, un conjunto adecuado de retrasos de tiempo y además es capaz de eliminar la tendencia de la serie [43].

Esta propuesta resuelve el problema de la tendencia con un pre y post procesado automático de la serie y mediante el uso de un algoritmo evolutivo (EA)

Dado que el objetivo principal del algoritmo implica construir al mismo tiempo los RBFNs y el conjunto de retrasos de tiempo, L-Co-R está basado en una aproximación coevolutiva. Esto es, el problema puede ser descompuesto en dos subproblemas codependientes uno del otro.

A continuación se presenta la composición del algoritmo L-Co-R

- Población de RBFNs: un conjunto de RBFNs que evolucionan para constituir el diseño apropiado de la red. La población usa una codificación real, en la que cada individuo representa un conjunto de neuronas (RBFs) que compone la red.
- Población de retrasos: conjunto de retrasos de tiempo evolucionados para predecir los valores futuros de cada serie. Esta población utiliza una codificación binaria donde cada gen representa si un retraso específico se emplea o no para realizar la predicción de la serie.

Unos individuos de cada población son por si mismos una posible solución al subproblema. La principal ventaja de L-Co-R es que es capaz de predecir cualquier serie temporal proporcionada para cualquier horizonte. El esquema general del algoritmo está disponible en [28].

Al final del proceso co-evolutivo, dos modelos formados por una red neuronal y un conjunto de retardos. El primer modelo se compone de la mejor red y su mejor colaborador. El segundo modelo, se compone del mejor conjunto de retardos y su mejor colaborador. Ambos modelos son evaluados nuevamente, para determinar cual de ellos ofrece un mejor comportamiento de predicción.

Una vez elegido el mejor modelo, se emplea para la predicción de los valores futuros en la serie temporal.

Por tanto, L-Co-R no es un método determinista, es por ello que se recomienda ejecutarlo para una misma serie temporal en varias ocasiones.

10.1.3 Weka Forecasting

Weka [42] es una herramienta desarrollada en Java como software libre por la Universidad de Waikato, New Zealand. Incluye una serie de algoritmos para el preprocesamiento y modelado de técnicas de minería de datos (clasificación, clustering, regresión y predicción).

El entorno de weka permite elegir el algoritmo o método de predicción que será realizado, incluyendo métodos no lineales bastante potentes como son las máquinas de vector soporte o SVM (Support Vector Machines) [39] para entrenar un modelo basado en perceptrón multicapa [10]; este tipo de métodos no lineales son mucho más potentes y flexibles para las tareas de predicción de series temporales frente a técnicas estadísticas como ARIMA. Incluye también modelos más sencillos como los basados en regresión lineal [25].

10.2 PREDICCIÓN POR DÍAS

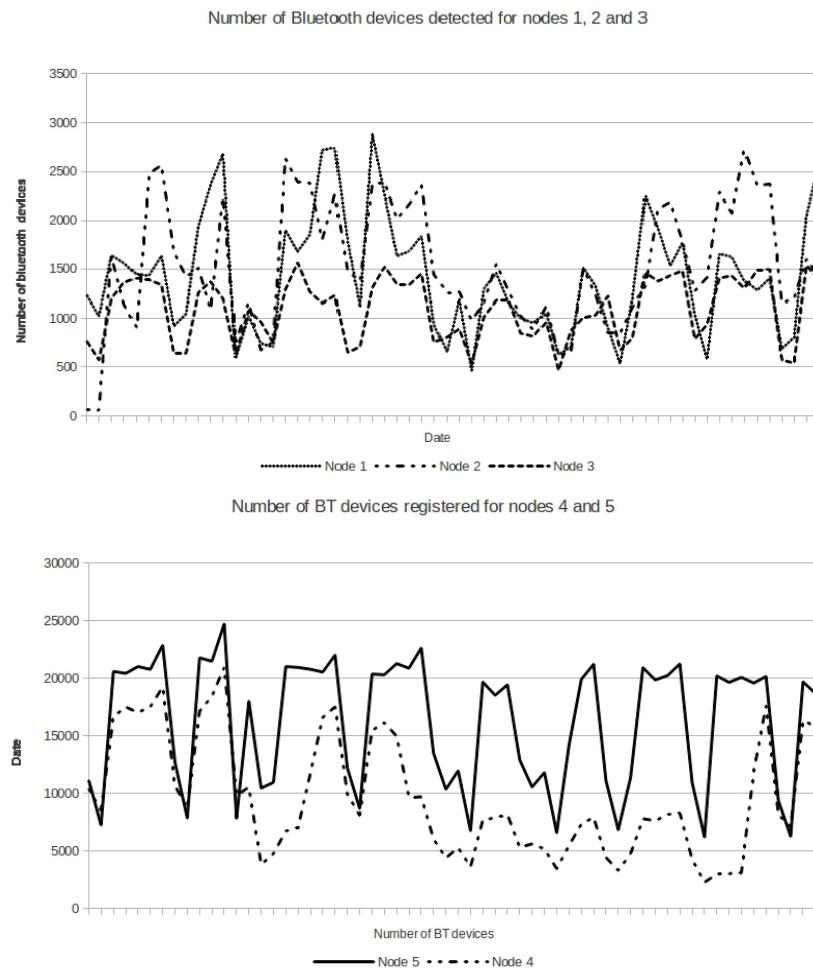
Los experimentos se llevaron a cabo con los datos recogidos durante 60 días. Los primeros 53 días son usados para el entrenamiento de los modelos predictivos. Los 7 restantes se usan para comprobar la bondad de las predicciones de cada uno de los modelos.

Como nodo, se emplearon 5 de los primeros nodos que fueron instalados en la fase de pruebas del hardware. La Figura 10.2 muestra el número de dispositivos detectados por cada nodo en cada día de la semana.

A partir de la serie temporal que conforma la serie temporal representada, se analiza el tráfico por día de la semana, quedando patente que el número total de dispositivos detectados claramente varía de los días laborables a los días no laborables (fines de semana principalmente). En la Figura 10.5 se ha acumulado el número de vehículos para los cinco nodos utilizados en estos experimentos; sin embargo, esta tendencia también se puede apreciar considerando un solo nodo.

En la Tabla 10.1 muestra los resultados en cuanto al error cometido con cada método probado. Se muestran los resultados

Figura 10.2
 Serie temporal de los nodos 1,2,3,4,5 con el número de vehículos detectado por día de la semana. Debido a la "mejor" posición de los nodos 4 y 5 han detectado mayor número de vehículos, por lo que se les presenta en una gráfica separada.



usando varias métricas de error tal y como se propone en [24] [35]. Así, para cada nodo, se calcula el error cometido con cada método probado, mostrando el error absoluto porcentual medio (Mean Absolute Percentage Error, MAPE [11]), error absoluto escalado medio (Mean Absolute Scaled Error, MASE [35]) y error cuadrático medio (mean squared error, MSE).

Como se puede observar, los mejores resultados se han obtenido con L-Co-R y ARIMA.

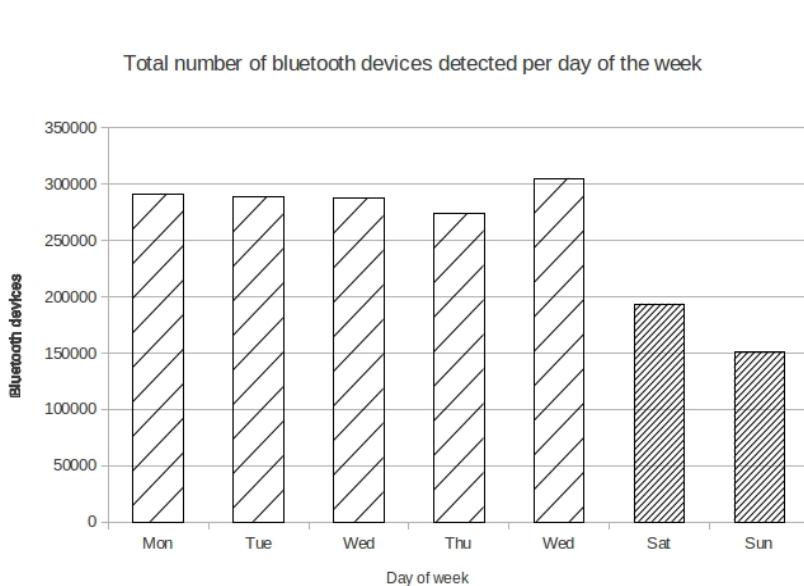


Figura 10.3
Total de dispositivos detectados para los cinco nodos empleados agrupados por día de la semana

Nodo 1			
Método	MAPE	MASE	MSE
ARIMA	40.42	1.14	424860.5
CROSTON	39.28	1.13	387403.9
THETA	41.16	1.17	443898.5
SPLINE	43.87	1.12	488504.9
L-Co-R	23.66	0.81	231731.8
Nodo 2			
Método	MAPE	MASE	MSE
ARIMA	26.64	1.03	310105.3
CROSTON	27.98	1.26	398979.9
THETA	26.77	1.03	312496.1
SPLINE	34.53	1.27	447557.4
L-Co-R	11.41	1.04	311561.7
Nodo 3			
Método	MAPE	MASE	MSE
ARIMA	35.11	1.32	150293.6
CROSTON	37.03	1.34	160970.9
THETA	38.11	1.4	175710.7
SPLINE	38.86	1.36	184807.2
L-Co-R	10.56	0.84	182469.2
Nodo 4			
Método	MAPE	MASE	MSE
ARIMA	40.62	1.04	21792093
CROSTON	56.83	1.25	25037291
THETA	39.45	1.01	21510757
SPLINE	39.45	1.24	28745309
L-Co-R	6.26	0.07	1120317
Nodo 5			
Método	MAPE	MASE	MSE
ARIMA	43.8	1.13	29974781
CROSTON	46.32	1.26	35082013
THETA	45.78	1.3	35859271
SPLINE	46.26	1.32	37288518
L-Co-R	24.59	0.87	35875329

Tabla 10.1

Métricas de error cometido por método en cada nodo. El mejor resultado de cada columna se muestra resaltado en negrita

10.3 PREDICCIÓN POR HORAS

Si bien la predicción respecto a series temporales basadas en días puede ser interesante en otros aspectos, la fluctuación del tráfico es demasiado variable respecto a un periodo de tiempo tan grande.

Es por ello que se decide empezar a trabajar con métodos de predicción usando las series temporales compuestas por los intervalos de horas del sistema. Así por ejemplo, resulta esperable que el tráfico disminuya de forma considerable durante las horas nocturnas y muestre un mayor afluencia de dispositivos capturados durante las horas diurnas.

Se consideran tres series con los resultados recogidos por 3 nodos de nuestro sistema durante las fechas del 31 de Diciembre del 2013 (00:00h) al 20 de Enero 2014 (23:59), agrupados en intervalos de horas. Se dispone por tanto de una serie de 505 valores, como se puede ver en la Figura 10.4.

Se usan algunas de las técnicas presentadas anteriormente para modelar un sistema que sea capaz de predecir las últimas 24 horas nuestra serie. Esto implica que para todos los métodos, los últimos 24 valores de la serie no son empleados para sus procesos de entrenamiento o constitución del modelo, siendo únicamente empleados estos últimos valores de la serie para la obtención de métricas de error sobre los valores predichos.

Como métricas de error [23], se emplean las siguientes. Se indica para cada una si es una métrica mayor-es-mejor o menor-es-mejor:

MAE Mean Absolute Error: Indica la media de los errores absolutos. Interesa por tanto un valor pequeño.

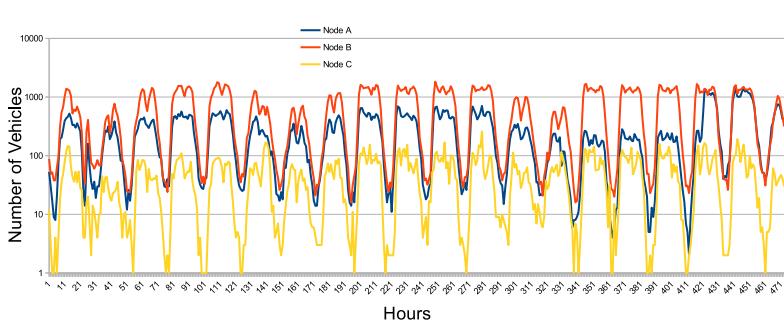


Figura 10.4
Serie temporal de los nodos 1,2,3 con el número de vehículos detectado por hora

RMSE Root mean squared error: Indica la raíz cuadrada de la media de los errores absolutos. Interesa por tanto un valor pequeño.

MAPE Mean absolute percentage error: Es la media de los errores absolutos en porcentaje, ya que el error medio va dividido por el elemento de la serie. Interesa por tanto un valor pequeño.

RRSE Root relative squared error: Obtenido como el cociente entre RMSE y el cálculo de RMSE si se hubiera predicho en cada paso el elemento anterior. Interesa por tanto un valor pequeño.

RAE Relative absolute error: Obtenido como el cociente entre MAE y la misma métrica de error MAE si se hubiera predicho el elemento anterior de la serie. Interesa por tanto un valor pequeño.

DA Direction accuracy: Indica el porcentaje de acierto de la predicción, siendo 100% un ajuste perfecto. Nos interesa por tanto un valor mayor.

MSE Mean squared error: Calculado como la suma de las diferencias al cuadrado normalizado respecto al tamaño de la serie. Nos interesa por tanto un valor pequeño.

Como métodos de predicción de los que se han descrito anteriormente se emplean:

- ARIMA
- SVM-MLP
- L-Co-R
- LR

Los resultados se muestran en la siguiente tabla:

Serie A	MAE	MAPE(%)	MSE	RMSE	RAE(%)	RRSE(%)	DA(%)
ARIMA	174,67	124,11	45751,58	213,90	218,81	214,34	54,17
SVM-MLP	211,72	78,96	90299,95	300,50	265,23	301,12	79,17
L-Co-R	288,36	84,81	146400,96	382,62	361,24	383,42	45,83
LR	57,37	51,64	5118,30	71,54	71,87	71,69	66,67

Serie B	MAE	MAPE(%)	MSE	RMSE	RAE(%)	RRSE(%)	DA(%)
ARIMA	420,50	433,13	259220,81	509,14	507,96	474,90	54,17
SVM-MLP	132,78	230,87	30287,99	174,03	160,39	162,33	58,33
L-Co-R	204,81	320,63	76271,78	276,17	247,41	257,60	54,17
LR	52,10	42,92	4571,18	67,61	62,94	63,06	58,33

Serie C	MAE	MAPE(%)	MSE	RMSE	RAE(%)	RRSE(%)	DA(%)
ARIMA	18,03	416,08	507,47	22,53	260,76	209,67	37,50
SVM-MLP	16,42	283,82	508,19	22,54	237,59	209,82	41,67
L-Co-R	17,48	183,07	571,37	23,90	252,84	222,48	54,17
LR	9,44	119,06	150,85	12,28	136,50	114,32	45,83

Tabla 10.2

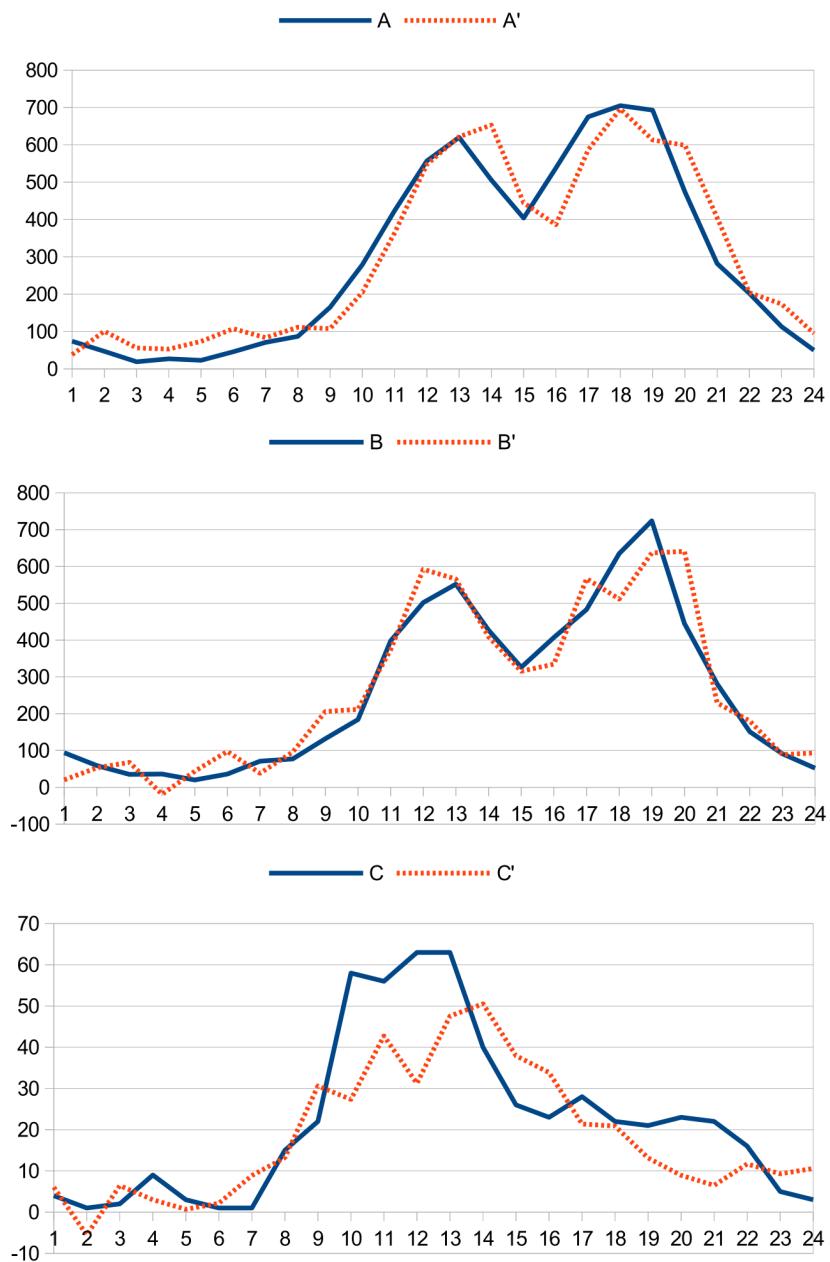
Métricas de error para las series temporales A, B y C. Se presenta en negrita el mejor método para cada métrica de error. El mejor valor es siempre el más bajo, salvo en el caso de la métrica DA, en cuyo caso es el mayor.

En la tabla se observa que el método que mejor resultados ofrece es *LR* (regresión lineal) que consigue para todas las series ofrecer las mejores métricas de error en al menos 6 de las 7 métricas.

En la siguiente figura observamos los valores reales de la series y los valores predichos.

Decidimos por tanto emplear para las predicciones *LR*, que además, es de los métodos más rápido de los que hemos probado para realizar las predicciones de forma online en nuestro sistema. Lo que nos da pie al siguiente capítulo.

Figura 10.5
Valores predichos de las series (rectas punteadas) comparadas con los valores reales de las series.



PREDICCIÓN ONLINE

ÍNDICE DE CAPÍTULO

- 11.1 Software empleado **126**
 - 11.1.1 R **126**
 - 11.1.2 Weka **128**
 - 11.2 SiMa: Módulo de Predicción **130**
 - 11.2.1 Carga de datos **130**
 - 11.2.2 Construcción del modelo de predicción **131**
 - 11.2.3 Controlador del Módulo **134**
-

En el anterior capítulo hemos presentado distintas técnicas para la predicción de series temporales, pero dichas técnicas han sido realizadas “offline”, es decir, con series temporales extraídas del sistema, y sin introducir sus valores predichos al sistema.

En este capítulo presentamos el módulo de predicción implementado para el sistema SiMa, que permite realizar predicciones “online”, lo que implica que las predicciones se realizan a medida que se recogen los datos en el sistema. Además, se adelantará el sistema de subida en la nube mediante fusion tables para su publicación. El módulo de subida de datos a la nube será presentado en el Capítulo 12. Para la comprensión de este capítulo sólo es necesario adelantar que dicho módulo dispone de métodos que permiten trabajar de forma nativa con las tablas alojadas en Google Fusion Tables, permitiendo la realización de funciones típicas de SQL.

11.1 SOFTWARE EMPLEADO

A la hora de hacer uso de cualquier técnica de predicción existente en la bibliografía, existen dos alternativas. La primera para por la implementación del método, el cual tiene que estar lo suficientemente documentado para poder permitir una reproducción de la implementación, lo cual no suele ser lo común, presentándose los métodos únicamente desde el punto de vista del modelo matemático.

La otra alternativa para por emplear algún paquete de software con implementaciones muy eficientes y probadas de métodos clásicos de predicción. En nuestro caso, se optó por probar dos paquetes de software para predicción.

11.1.1 R

R es un lenguaje de programación más que un entorno de software, dispone de innumerables librerías OpenSource para el procesamiento de información. En nuestro caso, nos interesamos por la librerías de predicción de series temporales basadas en métodos estadísticos (no en modelos entrenados) denominada *forecasting*.

Debido a ser un lenguaje externo, ofrecía escasa integración de forma nativa con nuestro sistema (desarrollado en JAVA).

Aún así, se realizó una versión preliminar del código que permite la ejecución de este entorno de predicción. En el siguiente código

se muestra un ejemplo de uso de los métodos de predicción ets y Holt Winters.

```

1 #Lectura de los datos
2 library(forecast)
3 data <- scan("/tmp/datosR.csv")
4 dataseries <- ts(data,start=c(1913))
5 png('/var/www/Rjava/t.png')
6
7 #Método de predicción ETS
8 forecast(ets(dataseries), 5)
9 plot( forecast(ets(dataseries), 5) )
10 png('/var/www/Rjava/t2.png')
11
12 #Método de predicción Holt Winters
13 rainseriesHW <- HoltWinters(rainseries, gamma=FALSE)
14 forecast.HoltWinters(rainseriesHW, h=5)
15 plot( forecast.HoltWinters(rainseriesHW, h=5) )

```

Figura 11.1
Código: Código para la predicción en R : Script en R

Este código toma los valores de la serie temporal desde un fichero de CSV almacenado en el fichero temporal. Veremos que esta manera de proceder fue salvada por una alternativa mucho más eficiente.

Para lanzar nuestro método de predicción basado en R desde nuestro sistema desarrollado en Java, se puede hacer uso del siguiente código:

```

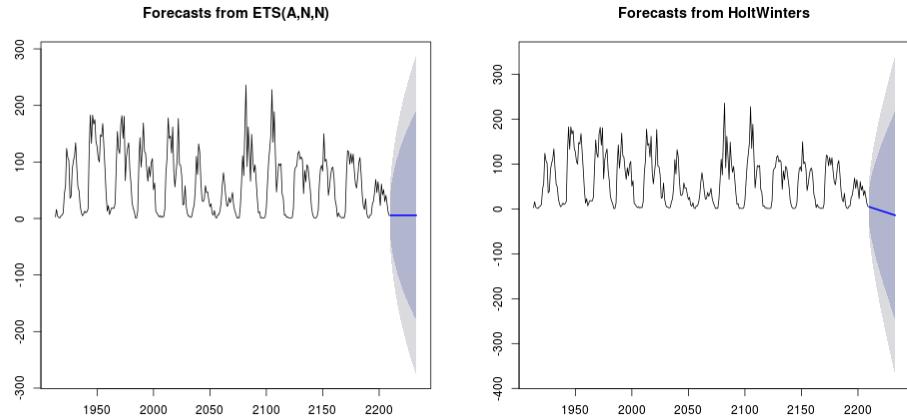
1 try {
2     //Ejecución del script
3     String rScriptFileName = localPath+"miscript.R";
4     Runtime.getRuntime().exec("/usr/bin/Rscript " + rScriptFileName);
5     //Espera a término de ejecución del script
6     try {Thread.sleep(10000);}
7     catch (InterruptedException e) {e.printStackTrace();}
8     //Lectura de la salida
9     String matchFileName = localPath+"resultado.csv";
10    BufferedReader br3 = new BufferedReader(new FileReader(matchFileName));
11    String thisRow;
12    int rowIndex = 0;
13    while ((thisRow = br3.readLine()) != null) {
14        System.out.println("Lin." + rowIndex + " -> " + "[" + thisRow + "]");
15        ;
16        rowIndex += 1;
17    }
18    br3.close();
19 } catch (FileNotFoundException e) {e.printStackTrace();}
20 catch (IOException ie){ie.printStackTrace();}

```

Figura 11.2
Código: Código para la predicción en R : Código fuente del módulo en Java.

Figura 11.3

Gráfica con serie temporal de pruebas, con la predicción realizada por los dos métodos en R



Este método tiene varias desventajas. La más evidente es que requiere de una llamada a software externo (el script en R) que no puede ser controlada con los mecanismos provistos en el sistema (como la depuración o la configuración).

Además, estos métodos estadísticos predicen la tendencia futura tal y como se muestra en la Figura 11.3, no los “valores” en el futuro.

Si bien este tipo de técnicas puede resultar útil para un análisis offline de los datos, su uso no parece el adecuado para un entorno Online.

11.1.2 Weka

Weka se describe como “un entorno para análisis del conocimiento”, ofreciendo para ello múltiples herramientas para el aprendizaje automático y la minería de datos. Aunque su forma más común es mediante un GUI, al ser OpenSource, ofrece también sus librerías como entorno para la utilización por programadores terceros.

Al estar programado en Java resulta un entorno excelente para integrarlo en nuestro sistema SiMa. Si bien las predicciones de series temporales no han sido soportadas (a modo de plugin externo) hasta la versión 3.7 de Weka (a la escritura de este documento aún en estado de “desarrollo”), sus librerías resultan muy útiles para la utilización de métodos de predicción dentro de nuestro sistema.

A continuación un ejemplo de uso de predicción haciendo uso de las librerías de Weka:

```
1 import java.io.*;
2 import java.util.List;
3 import weka.core.Instances;
4 import weka.classifiers.functions.GaussianProcesses;
5 import weka.classifiers.evaluation.NumericPrediction;
6 import weka.classifiers.timeseries.WekaForecaster;
7 import weka.classifiers.timeseries.core.TSLagMaker;
8
9 public class TimeSeriesExample {
10
11     public static void main(String[] args) {
12         try {
13             String pathToWineData = weka.core.WekaPackageManager.PACKAGES_DIR.
14                 toString()
15             + File.separator + "timeseriesForecasting" + File.separator + "
16                 sample-data"
17             + File.separator + "wine.arff";
18
19             Instances wine = new Instances(new BufferedReader(new FileReader(
20                 pathToWineData)));
21
22             WekaForecaster forecaster = new WekaForecaster();
23
24             forecaster.setFieldsToForecast("Fortified,Dry-white");
25
26             forecaster.setBaseForecaster(new GaussianProcesses());
27
28             forecaster.getTSLagMaker().setTimeStampField("Date");
29             forecaster.getTSLagMaker().setMinLag(1);
30             forecaster.getTSLagMaker().setMaxLag(12);
31
32             forecaster.getTSLagMaker().setAddMonthOfYear(true);
33
34             forecaster.getTSLagMaker().setAddQuarterOfYear(true);
35
36             forecaster.buildForecaster(wine, System.out);
37
38             forecaster.primeForecaster(wine);
39
40             List<List<NumericPrediction>> forecast = forecaster.forecast(12,
41                 System.out);
42
43             for (int i = 0; i < 12; i++) {
44                 List<NumericPrediction> predsAtStep = forecast.get(i);
45                 for (int j = 0; j < 2; j++) {
46                     NumericPrediction predForTarget = predsAtStep.get(j);
47                     System.out.print(" " + predForTarget.predicted() + " ");
48                 }
49                 System.out.println();
50             }
51         } catch (Exception ex) {
52             ex.printStackTrace();
53         }
54     }
55 }
```

Figura 11.4
Código: Código para la predicción en Weka: Ejemplo.

11.2 SIMA: MÓDULO DE PREDICCIÓN

De igual manera que con módulos anteriores, en esta sección se presentan los componentes encargados de la predicción para posteriormente presentar el módulo encargado del correcto funcionamiento y sincronización de todos los componentes.

11.2.1 *Carga de datos*

En los anteriores ejemplos presentados se hacía uso de ficheros temporales para la carga de los datos al modelo de predicción a entrenar. Esto es debido a que las librerías de WEKA sólo ofrecen mecanismos para cargar de forma nativa Instancias desde ficheros ARFF.

Sin embargo, el tener que realizar este volcado a disco constantemente para realizar la predicción no resulta una situación favorable, debido a que el fichero en disco puede ser modificado sin que el módulo tenga constancia de ello. Es por ello que se implementa un método `cargarDatos` que sirve para crear un conjunto de `Instancias` con la que las librerías de Weka pueden trabajar de forma nativa mediante una petición a nuestra base de datos en la nube (Google Fusion Tables). El hecho de emplear nuestra base de datos en la nube frente a la base de datos local, es para interferir lo menos posible en los métodos de procesamiento de datos de nuestra base de datos.

A continuación se presenta el código encargado de realizar la carga desde una petición a nuestra base de datos en la Nube (Google Fusion Tables). En dicho código se adelanta el módulo de actualizador en la nube que será presentado en el Capítulo 12 tal y como se ha indicado al principio de este capítulo. Lo único que es necesario saber sobre el método `cFT.select` es que nos permite trabajar con nuestra base de datos en la nube como con cualquier otra base de datos¹.

¹ Dicho método forma parte de la implementación de nuestro sistema de sincronización en la nube, lo mismo que la clase `conectarFusiontables` por lo que se hayan documentadas sólo en este documento y en documentos relativos al proyecto SIEPsCA

```

1 ArrayList<Instances> cargarDatos() throws ParseException {
2     //Declaramos los atributos de las instancias
3     Attribute a0 = new Attribute("Intervalo", "yyyy-MMdd HHmm:ss");
4     Attribute a1 = new Attribute("Total");
5     ArrayList<Attribute> c = new ArrayList<>(); c.add(a0); c.add(a1);
6     //Creamos el conjunto de instancias
7     ArrayList<Instances> instances = new ArrayList<>(24);
8     for (int i = 0; i < 24; i++) {instances.add(new Instances(nodo, c, 1000)
9         );}
10
11     //Instanciamos conexión con FT
12     CFT = new conectarFusionTables();
13     Sqlresponse r = CFT.select(TABLAID, "Intervalo, Total,polygono", "idNodo
14         = " + nodo + " AND Total>1 ", "ORDER BY `Intervalo` DESC LIMIT
15         10000");
16     primera_fecha = (String) r.getRows().get(0).get(0);
17     for (List<Object> a : r.getRows()) {
18         Instance i = new DenseInstance(2);
19         String s0 = (String) a.get(0);
20         if (!s0.equals(ultima_fecha)) {
21             String s1 = (String) a.get(1);
22             int hora = Integer.parseInt(s0.substring(11, 13));
23             i.setValue(instances.get(hora).attribute(0), instances.get(hora)
24                 .attribute(0).parseDate(s0));
25             i.setValue(instances.get(hora).attribute(1), Integer.parseInt(s1
26                 ));
27             instances.get(hora).add(i);
28         }
29         ultima_fecha = (String) a.get(0);
30     }
31     for (Instances a : instances) {a.sort(0);}
32     return instances;
33 }
```

Figura 11.5

Código: Método que carga datos desde una petición a la base de Datos Fusion y los convierte en un formato soportado por las librerías de Weka

11.2.2 Construcción del modelo de predicción

Cómo técnica de predicción empleada en este módulo usamos un método muy simple como es la regresión lineal. Dado a que estamos en un entorno online, es de vital importancia que el método sea muy rápido frente a la bondad de dicho método.

Además, debido a que queremos hacer predicciones de intervalos de tiempo “cortos”, empleamos un método de considerar la serie de forma distinta. En este ejemplo que presentamos aquí, queremos hacer predicciones cada hora. Debido a la estacionalidad de los datos, el hacer una serie temporal con todos los registros de un nodo concreto sería una serie demasiado grande, por lo que el cómputo asociado sería mayor. En lugar de formar una serie temporal con todos los registros del nodo, hacemos una serie temporal sólo con los registros de una hora determinada.

Así, por ejemplo si queremos hacer una predicción del número de dispositivos esperables a ser detectados en un nodo en un intervalo concreto (por ejemplo, las de las 14 a 15 horas), formamos una serie temporal con todos los valores de dispositivos detectados por nuestro sensor en el dicho intervalo a lo largo del tiempo. De esta manera, podemos hacer uso de un histórico de datos sin hacer uso de todos los datos almacenados.

Además se ha realizado así al considerar que puede influir más en la predicción del tráfico el estado del tráfico en ese mismo intervalo de tiempo a lo largo de varios días y semanas, que frente a los valores de todo el día de la última semana.

Sin embargo, es necesario notar que este sistema se encuentra aún en una fase de desarrollo muy experimental, siendo la alternativa aquí mostrada, la que mejor al cierre de este documento.

Se presenta por tanto el código asociado a la obtención del modelo de predicción:

```

1 public void run() {
2     try {
3         ArrayList<Instances> pasos = cargarDatos();
4         //Instanciamos el predictor
5         ArrayList<WekaForecaster> forecaster = new ArrayList<>(24);
6         for (int a = 0; a < 24; a++) {forecaster.add(new WekaForecaster());}
7         int a = 0;
8         Date fecha = Debug.sdf.parse(primeria_fecha);
9         Calendar cal = Calendar.getInstance();
10        cal.setTime(fecha);
11        for (WekaForecaster fore : forecaster) {
12            cal.add(Calendar.HOUR, 1);
13            //Definimos el atributo que queremos predecir
14            fore.setFieldsToForecast("Total");
15            //Definimos el método de predicción a emplear. En este caso,
16            //regresión lineal porque
17            //en el artículo es el que mejor ha funcionado
18            fore.setBaseForecaster(new LinearRegression());
19            //Definimos el atributo que "marca" el tiempo y su peridiocidad
20            fore.getTSLagMaker().setTimeStampField("Intervalo");
21            fore.getTSLagMaker().setMinLag(1);
22            fore.getTSLagMaker().setMaxLag(1);
23            fore.getTSLagMaker().setPeriodicity(TSLagMaker.Periodicity.WEEKLY);
24            fore.buildForecaster(pasos.get(a), System.out);
25            fore.primeForecaster(pasos.get(a));
26            List<List<NumericPrediction>> forecast = fore.forecast(1, System.out
27                );
28
29            for (int i = 0; i < 1; i++) {
30                List<NumericPrediction> predsAtStep = forecast.get(i);
31                for (int j = 0; j < 1; j++) {
32                    NumericPrediction predForTarget = predsAtStep.get(j);
33                    List<String> valores = new ArrayList<>();
34                    valores.add(Debug.sdf.format(cal.getTime()));valores.add(
35                        this.nodo);
36                    valores.add("");valores.add(Double.toString(predForTarget.
37                        predicted()));
38                    valores.add(poligono);
39                    CFT.insert(TABLAID, campos, valores, true, 2);
40                }
41            }
42        }
43    }
44    CFT.forzarSync();
45    CFT.esperarSubida();
46 }
```

Figura 11.6

Código: Método que realiza la predicción para el nodo solicitado de sus futuros valores

En dicho código, figura nuevamente una función asociada al módulo de publicación en la nube, en este caso, dicha función es la encargada de añadir (o actualizar) la predicción en nuestras nube con Google Fusion Tables.

11.2.3 Controlador del Módulo

En los apartados anteriores hemos mostrado las distintas partes que constituyen nuestro módulo de predicción: el submódulo encargado de la carga de datos desde Google Fusion Tables y el módulo encargado de realizar la predicción y publicación del resultado predicho.

El controlador del módulo de predicción se basa en la ejecución de una tarea programada que realiza la ejecución en hebras de la predicción. Se puede observar su funcionamiento en el siguiente código.

```
1 public void run() {
2     this.temporizador = new TimerTask() {
3
4         public void run() {
5             Logger.getGlobal().log(Level.INFO, "Comenzando sincronizado
6                 programado en la nube de la predicción.");
7             System.err.println("Comenzando sincronizado programado en la nube de
8                 la predicción.");
9             System.err.println("————→" + clientNo.static_getHowManyNodos());
10
11            for (int i = 0; i < clientNo.static_getHowManyNodos(); i++) {
12                try {
13                    hebras.add(new Prediccion(clientNo.static_getNodo(i)));
14                    hebras.peek().start();
15                    hebras.peek().join();
16                    hebras.poll();
17                } catch (InterruptedException ex) {
18                    Logger.getLogger(ActualizadorPrediccion.class.getName()).log
19                        (Level.SEVERE, null, ex);
20                }
21            }
22        }
23    };
24}
```

Figura 11.7

Código: Tarea temporizada de predicción del Actualizador de Predicción

Debido a estar una fase temprana y experimental, aún no se hace un uso exhaustivamente paralelo de las hebras, realizando el procesamiento secuencialmente. Este modo de funcionamiento es muy más sencillo de depurar, aunque el sistema está preparado para la realización de la tarea de predicción en paralelo para cuando sea necesario.

En función de lo requerido por el fichero de configuración, el sistema realiza esta tarea programada a lo largo del tiempo, tal y como se muestra en el siguiente código.

```
1 public void start(){  
2     Timer timer = new Timer("ActualizadorPredicción", true);  
3     timer.scheduleAtFixedRate(temporizador, _c.getLong("ft.  
        predicción_tiempo_espera"),_c.getLong("ft.  
        predicción_período_actualización"));  
4 }
```

Figura 11.8

Código: Instanciación de la tarea temporizada de predicción del Actualizador de Predicción

Parte V

PUBLICACIÓN Y DIFUSIÓN EN LA NUBE

12

SIMA: ACTUALIZADOR EN LA NUBE CON GOOGLE FUSION TABLES

ÍNDICE DE CAPÍTULO

- 12.1 Autorización **140**
 - 12.2 Optimización de procesado: Sistema de colas **141**
 - 12.3 Métodos desarrollados para trabajar con métodos SQL **144**
 - 12.3.1 Métodos de selección: SELECT **145**
 - 12.3.2 Métodos de inserción: INSERT **145**
 - 12.3.3 Métodos de actualización: UPDATE **151**
 - 12.3.4 Métodos de borrado: DELETE **151**
 - 12.4 Sistemas de Actualización **152**
 - 12.4.1 Actualización de Nodos **152**
 - 12.4.2 Actualización de Pasos por Horas **153**
 - 12.4.3 Actualización de Trazas por Horas **155**
 - 12.5 Cliente Actualizador FT **157**
-

En este capítulo se describe el módulo *Actualizador Fusion Tables* desarrollado para nuestro sistema SiMa con el fin de poder trabajar de forma nativa con las bases de datos alojadas en Google Fusion Tables. Dicho módulo ha sido comentado con anterioridad a lo largo de este documento, debido a su importancia en otros módulos. Si bien se considera el módulo de *Actualización Local* prioritario, este módulo sería el segundo módulo principal del sistema, pues es el encargado de “hacer visible” la información adquirida y procesada.

Partimos de la librería desarrollada por Google para trabajar con su API en el entorno JAVA, ampliéndola para hacer el funcionamiento mucho más cómodo y sencillo. Se desarrolla la clase `ConecitarFusionTables` como interfaz para realizar todas las conexiones con Google Fusion Tables.

12.1 AUTORIZACIÓN

Si bien el acceso de lectura de los datos puede ser público, la escritura y modificación de los datos asociados a un usuario de Google Fusion Tables requiere un mecanismos de autenticación, que identifique a un usuario con permisos de escritura frente a un usuario sin ningún tipo de privilegios.

Google hace uso del popular servicio de autenticación OAuth, que permite la autorización de una API de modo estándar y simple. La gestión del modo de autenticación requiere el almacenaje y envío de credenciales de usuarios, basadas en un sistema de clave pública / privada [2][1].

===== [Añadir referencias] =====

<https://developers.google.com/accounts/docs/OAuth2>

O para la siguiente otra dirección para ver el empleo y gestión de los permisos en la plataforma Google Fusion Tables:

<https://developers.google.com/fusiontables/docs/v1/using#auth>

Realizamos la autorización de nuestro módulo con la siguiente función:

```
1 private Credential authorize() throws Exception {
2     FileInputStream _f = new FileInputStream(DATA_STORE_FILE);
3
4     GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(
5         JSON_FACTORY,
6         new InputStreamReader(_f));
7
8     if (clientSecrets.getDetails().getClientId().startsWith("Enter") ||
9         clientSecrets.getDetails().getClientSecret().startsWith("Enter "))
10    ) {
11        Logger.getGlobal().log(Level.SEVERE,
12            "Enter Client ID and Secret from https://code.google.com/apis/
13            /console/?api=fusiontables "
14            + "into fusiontables-cmdline-sample/src/main/resources/
15            client_secrets.json");
16        System.exit(1);
17    }
18
19    GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.
20        Builder(
21            httpTransport, JSON_FACTORY, clientSecrets,
22            Collections.singleton(FusiontablesScopes.FUSIONTABLES)).
23                setDataStoreFactory(
24                    dataStoreFactory).build();
25
26    return new AuthorizationCodeInstalledApp(flow, new LocalServerReceiver
27        ()).authorize("user");
28 }
```

Figura 12.1

Código: Código para la gestión de credenciales de conexión a Google Fusion Tables

Dicho código hace uso de una constante DATA_STORE_FILE que es inicializada en el constructor de la clase para abrir el fichero de credenciales, usando el módulo de configuración presentado en el Capítulo 6.

12.2 OPTIMIZACIÓN DE PROCESADO: SISTEMA DE COLAS

La utilización de una API para la subida de datos, como es el caso que nos ocupa, corre el riesgo de sobrepasar la cuota de usuario de la que se disponga. De igual manera que para gestionar las inserciones en la base de datos local, se realiza un sistema de colas para gestionar las subidas.

En este caso, no se implementa como un mecanismo de paralelismo, sino como mecanismo de serialización. Esto puede resultar curioso, pero es preferible realizar la subida de datos poco a poco con el fin de no sobrepasar la cuota de subida de la API.

Además, de esta forma no saturamos la salida a internet, disponiendo de todo el ancho de banda para una única subida de datos.

Hacemos uso de una lista de estructura de tipo cola para almacenar los datos que son necesario enviar.

```
1  /**
2   * Variable que almacena la query insert cacheada
3   */
4  private String insertCache = "";
5  /**
6   * Variable de caché de la caché (¡Cómo en origen!)
7   */
8  private static Queue<String> insertCacheLista = new LinkedList<>();
9  /**
10  * Máximo de INSERTS a almacenar en la cache
11  */
12  private final int INSERT_MAX_CACHE = _c.getInt("ft.insert_cache_size");
13  /**
14  * Variable que indica el tamaño actual de la caché de procesamiento de
15  * inserts
16  */
17  private int insertCacheContador = 0;
18  /**
19  * Manejador de colas de peticiones a FusionTables
20  */
21  private static colasManejadorInsert _colas = new colasManejadorInsert();
```

Figura 12.2

Código: Código para la gestión de colas para Fusion Tables I

Necesitamos por tanto un manejador para vaciar la cola de peticiones a medida que se vayan procesando las peticiones. Este manejador se presenta en el siguiente código:

```

1 /**
2  * Clase encargada de manejar las colas de peticiones de insercción a
3  * FusionTable
4 */
5 private static class colasManejadorInsert extends Thread {
6     static String pendienteProcesar = null;
7     Boolean vacio = false;
8
9     public colasManejadorInsert() {
10         this.setName("Manejador colas Inserts FT");
11     }
12     @Override
13     public void run() {
14         do {
15             synchronized (insertCacheLista) {
16                 Logger.getGlobal().fine("Soy la hebra manejadora de FT " +
17                     insertCacheLista.size() + " elementos pendientes");
18                 if (insertCacheLista.isEmpty()) {
19                     try {
20                         insertCacheLista.wait();
21                         pendienteProcesar = insertCacheLista.poll();
22                     } catch (InterruptedException ex) {
23                         Logger.getGlobal().log(Level.SEVERE, null, ex);
24                     }
25                 } else if (pendienteProcesar == null || "".equals(
26                     pendienteProcesar)) {
27                     pendienteProcesar = insertCacheLista.poll();
28                 }
29             }
30             Logger.getGlobal().fine("Soy la hebra manejadora y voy a procesar " +
31                     + pendienteProcesar);
32             if (sqlStatic(pendienteProcesar) == null) {
33                 try {
34                     //Ha fallado la transacción, por lo que la tenemos que
35                     //volvemos a procesar pasados unos segundos
36                     sleep(_c.getInt("ft.tiempo_espera_error_ms"));
37                 } catch (InterruptedException ex) {
38                     Logger.getGlobal().log(Level.SEVERE, null, ex);
39                 }
40             } else {
41                 //Se ha procesado correctamente
42                 pendienteProcesar = null;
43             }
44         } while (true);
45     }
46 }

```

Figura 12.3
Código: Código para la gestión de colas para Fusion Tables II

Este controlador se encarga de ir enviando los paquetes de datos pendientes a Google Fusion Tables. Es necesario notar que este mecanismo se emplea solamente para las inserciones de nuevos datos. Las consultas y modificaciones son gestionadas fuera de este entorno, debido a que se las considera más urgente, es decir, requieren una respuesta inmediata para continuar la ejecución.

12.3 MÉTODOS DESARROLLADOS PARA TRABAJAR CON MÉTODOS SQL

Por defecto, la API de Fusion Table sólo permite enviar un tipo genérico de QUERY. Para facilitar el tratamiento de los datos, se implementan métodos que permitan de forma nativa trabajar con las operaciones más frecuentes en una base de datos: SELECT, INSERT, UPDATE, DELETE.

Salvo las inserciones todas ellas hacen uso del método básico SQL que se presenta a continuación:

```

1  /** Procesa una petición SQL básica
2   * @param query cadena de texto que indica la petición SQL a realizar
3   * @return El archivo JSON procesado devuelto por la petición */
4  private Sqlresponse sql(String query) {
5      if (query == "" || query == null) { return null; }
6      Sqlresponse res = null;    boolean salir = false;
7      while (!salir) {
8          try {
9              Logger.getGlobal().fine("Procesando:" + query);
10             Sql sql = fusiontables.query().sql(query);
11             res = sql.execute(); salir = true;
12         } catch (Exception ex) {
13             if (ex.getMessage().contains("403") || ex.getMessage().contains("Read
timed out")) {
14                 try {
15                     sleep(_c.getInt("ft.tiempo_espera_error_ms"));
16                 } catch (InterruptedException ex1) {
17                     Logger.getGlobal().log(Level.SEVERE, "Error al dormir la hebra de
Procesado de subidas a la nube", ex1); }
18                     salir = false;
19                     Logger.getGlobal().fine("Envío fallido " + ex.getMessage() + " .
Demasiado rápido. Se volverá a intentar el envío");
20                 } else if (ex.getMessage().contains("503")) {
21                     //Hemos excedido la cuota
22                     salir = false;
23                     Logger.getGlobal().fine("Envío fallido " + ex.getMessage() + " .
Cuota excedida. Se volverá a intentar el envío pasados " + _c.
getInt("ft.tiempo.esperaSubida.dormir") / 1000 + " segundos.");
24                     try {
25                         sleep(_c.getInt("ft.tiempo.esperaSubida.dormir"));
26                     } catch (InterruptedException ex1) {
27                         Logger.getGlobal().log(Level.SEVERE, "Error al dormir la hebra de
Procesado de subidas a la nube", ex1); }
28                 } else {
29                     salir = false;
30                     Logger.getGlobal().log(Level.SEVERE, "Envío fallido " + ex.
getMessage() + "" + query + "", ex); }
31             }
32         }
33     Logger.getGlobal().fine("Procesado correctamente.");
34     return res;
35 }
```

Figura 12.4
Código: Métodos para Fusion Table: SQL

12.3.1 *Métodos de selección: SELECT*

Se han implementado varias operaciones de SELECCION obedeciendo a la necesidad de sobrecargar el método en función de los parámetros pasados. Se presenta a continuación los dos métodos de selección:

```
1  /**
2  * Función que realiza una selección de una tabla
3  *
4  * @param tabla Identificador de la tabla que se quiere consultar
5  * @param campos Campos que queremos recuperar
6  * @param condiciones Condiciones que tiene que cumplir las tuplas para
7  *                   que
8  *                   sean devueltas
9  * @return
10 */
11 public Sqlresponse select(String tabla, String campos, String condiciones
12 ) {
13     return sql("SELECT " + campos + " FROM " + tabla + " WHERE " +
14             condiciones);
15 }
16 /**
17 * Función que realiza una selección de una tabla con condiciones extras
18 *
19 * @param tabla Identificador de la tabla que se quiere consultar
20 * @param campos Campos que queremos recuperar
21 * @param condiciones Condiciones que tiene que cumplir las tuplas para
22 *                   que
23 *                   sean devueltas
24 * @return
25 */
26 public Sqlresponse select(String tabla, String campos, String condiciones
27 , String extras) {
28     if (!"".equals(condiciones)) {
29         return sql("SELECT " + campos + " FROM " + tabla + " WHERE " +
30                 condiciones + " " + extras);
31     } else {
32         return sql("SELECT " + campos + " FROM " + tabla + " " + extras);
33     }
34 }
```

Figura 12.5
Código: Métodos para Fusion Table: SELECT

12.3.2 *Métodos de inserción: INSERT*

De igual manera que con la selección, sobrecargamos el método para permitir invocarlo con distintos parámetros. En la siguiente figura se encuentran los métodos sobrecargados.

```

1  /**
2  * Sobre carga del método insert para no indicar que se compruebe que
3  * existe
4  * el elemento. Por defecto, el elemento a insertar se comprueba si
5  * existe
6  * en la base de datos de FUSION TABLES antes de insertarlo
7  *
8  * @param tabla identificador de la tabla
9  * @param campos listado de campos (separados por comas) de la tabla
10 * @param valores listado de valores (separados por comas) a insertar
11 * @return La respuesta devuelta por el servidor de Fusion Tables.
12 */
13 public Sqlresponse insert(String tabla, List<String> campos, List<String>
14                           valores) {
15     return insert(tabla, campos, valores, true);
16 }
17 /**
18 * Sobre carga del método insert para no indicar que se compruebe que
19 * existe
20 * el elemento y para un solo elemento. Por defecto, el elemento a
21 * insertar
22 * se comprueba si existe en la base de datos de FUSION TABLES antes de
23 * insertarlo
24 *
25 * @param tabla identificador de la tabla
26 * @param campos listado de campos (separados por comas) de la tabla
27 * @param valores listado de valores (separados por comas) a insertar
28 * @return La respuesta devuelta por el servidor de Fusion Tables.
29 */
30 public Sqlresponse insert(String tabla, String campos, String valores) {
31     List<String> c = new ArrayList<>();
32     List<String> v = new ArrayList<>();
33
34     c.add(campos);
35     v.add(valores);
36
37     return insert(tabla, c, v, true);
38 }
39 /**
40 * Sobre carga del método insert para un solo elemento.
41 *
42 * @param tabla identificador de la tabla
43 * @param campos listado de campos (separados por comas) de la tabla
44 * @param valores listado de valores (separados por comas) a insertar
45 * @return La respuesta devuelta por el servidor de Fusion Tables.
46 */
47 public Sqlresponse insert(String tabla, String campos, String valores,
48                           boolean check) {
49     List<String> c = new ArrayList<>();
50     List<String> v = new ArrayList<>();
51
52     c.add(campos);
53     v.add(valores);
54
55     return insert(tabla, c, v, check);
56 }

```

Figura 12.6

Código: Métodos para Fusion Table: INSERT Sobre carga

La inserción implica la operación más compleja realizable en nuestras bases de datos Fusion Tables. Esto es debido a que al carecer de clave primaria establecida por nosotros, podemos correr el riesgo de subir varias veces la misma tupla. Al carecer de un mecanismo como el `ON DUPLICATE KEY UPDATE` que proporciona SQL, antes de realizar una inserción debemos comprobar que la tupla que deseamos insertar no se encuentra ya insertada. Es por ello que se debe realizar una función de selección con toda la tupla (o parte de ella), y comprobar si existe ya almacenada en nuestro sistema.

Debido a la ineficiencia de manera de proceder, (deben de hacerse el doble comunicaciones para insertar cada tupla), se permite no realizar dicha comprobación. Este comportamiento se parametriza con el valor `check` que se muestra en el código anterior.

```

1  /**
2   * Función que inserta una nueva tupla en la tabla alojada en FusionTables
3   *
4   * @param tabla identificador de la tabla
5   * @param campos listado de campos (separados por comas) de la tabla
6   * @param valores listado de valores (separados por comas) a insertar
7   * @param check si es necesario realizar comprobación de insercción para en
8   * caso de ocurrencia usar UPDATE
9   * @return La respuesta devuelta por el servidor de Fusion Tables.
10  */
11 public Sqlresponse insert(String tabla, List<String> campos, List<String>
12                           valores, boolean check) {
13     String peticion;
14     if (check) {
15         peticion = "SELECT ROWID FROM " + tabla + " WHERE ";
16         for (int i = 0; i < campos.size(); i++) {
17             if (i != 0) { peticion = peticion + " AND "; }
18             peticion = peticion + campos.get(i) + "=" + valores.get(i) + "\'";
19         }
20         Sqlresponse s = this.sql(peticion);
21
22         if (s.size() == 2) {
23             peticion = "INSERT INTO " + tabla + "(";
24             for (int i = 0; i < campos.size(); i++) {
25                 if (i != 0) {
26                     peticion = peticion + ",";
27                 }
28                 peticion = peticion + campos.get(i);
29             }
30             peticion = peticion + ") VALUES (";
31             for (int i = 0; i < valores.size(); i++) {
32                 if (i != 0) { peticion = peticion + ","; }
33                 peticion = peticion + "\'" + valores.get(i) + "\'";
34             }
35             peticion = peticion + ")";
36             insertCache = insertCache + peticion;
37             insertCacheContador++;
38         } else {
39             return this.update(tabla, campos, valores, s.getRows());
40         }
41     } else {
42         peticion = "INSERT INTO " + tabla + "(";
43         for (int i = 0; i < campos.size(); i++) {
44             if (i != 0) { peticion = peticion + ","; }
45             peticion = peticion + campos.get(i);
46         }
47         peticion = peticion + ") VALUES (";
48         for (int i = 0; i < valores.size(); i++) {
49             if (i != 0) { peticion = peticion + ","; }
50             peticion = peticion + "\'" + valores.get(i) + "\'";
51         }
52         peticion = peticion + ")";
53         insertCache = insertCache + peticion;
54         insertCacheContador++;
55     }
56     sync();
57     return null;
}

```

Figura 12.7
Código: Métodos para Fusion Table: INSERT

Sin embargo, este método comprueba todas las columnas de la tupla para ver si la tupla se encuentra ya en la base de datos o no. Es por ello que se realiza el siguiente método ampliado para permitir decidir el número de columnas que son empleadas en la comprobación de que la tupla ya se encuentra en el sistema.

```

1  /** @param para número de campos de la lista valores a utilizar en la
2   * comparación en caso de que check sea verdadero */
3  public Sqlresponse insert(String tabla, List<String> campos, List<String>
4      valores, boolean check, int para) {
5      String peticion;
6      if (check) {
7          peticion = "SELECT ROWID ";
8          for (int i = para; i < campos.size(); i++) {
9              peticion = peticion + "," + campos.get(i);
10         }
11         peticion = peticion + " FROM " + tabla + " WHERE ";
12         for (int i = 0; i < para; i++) {
13             if (i != 0) { peticion = peticion + " AND "; }
14             peticion = peticion + campos.get(i) + "=" + valores.get(i) + "=";
15         }
16         Sqlresponse s = this.sql(peticion);
17         if (s.size() == 2) {
18             peticion = "INSERT INTO " + tabla + "(";
19             for (int i = 0; i < campos.size(); i++) {
20                 if (i != 0) { peticion = peticion + ","; }
21                 peticion = peticion + campos.get(i);
22             }
23             peticion = peticion + ") VALUES (";
24             for (int i = 0; i < valores.size(); i++) {
25                 if (i != 0) { peticion = peticion + ","; }
26                 peticion = peticion + "\'" + valores.get(i) + "\'";
27             }
28             peticion = peticion + ");";
29             insertCache = insertCache + peticion; insertCacheContador++;
30         } else {
31             boolean cambio = false; int j = 1; String depura = "";
32             for (int i = para; (i < campos.size()) && cambio==false; i++) {
33                 try{
34                     if (!valores.get(i).equals((String) s.getRows().get(0).get(j))) {
35                         cambio = true;
36                         Logger.getLogger().fine("Son distintos " + depura);
37                     }catch(ClassCastException ex){
38                         Object v = s.getRows().get(0).get(j);
39                         if (!valores.get(i).equals(v.toString())) {
40                             cambio = true;
41                         }
42                     }catch(Exception ex){ cambio = true; }
43                     j++; //En 0 está ROWID
44                 }
45                 if (cambio == true) {
46                     return this.update(tabla, campos, valores, s.getRows());
47                 } else { return null; }
48             }
49         } else {
50             peticion = "INSERT INTO " + tabla + "(";
51             for (int i = 0; i < campos.size(); i++) {
52                 if (i != 0) {peticion = peticion + ","; }
53                 peticion = peticion + campos.get(i);
54             }
55             peticion = peticion + ") VALUES (";
56             for (int i = 0; i < valores.size(); i++) {
57                 if (i != 0) { peticion = peticion + ","; }
58                 peticion = peticion + "\'" + valores.get(i) + "\'";
59             }
60             peticion = peticion + ");";
61             insertCache = insertCache + peticion;
62             insertCacheContador++;
63         }
64         sync();
65     }
}

```

Figura 12.8
Código: Métodos para Fusion Table: INSERT II

12.3.3 *Métodos de actualización: UPDATE*

Esta función (que ya se ha mostrado su uso en el método insertar), permite actualizar la información de una tupla concreta. Para ello, se requiere proporcionar el ROWID o identificador de tupla, que puede ser adquirido realizando una selección.

```

1  /**
2  * Función que actualiza una tupla en la tabla alojada en FusionTables
3  *
4  * @param tabla identificador de la tabla
5  * @param campos campo que será actualizado
6  * @param valores valor que será actualizado
7  * @param ROWID identificador de la tupla
8  * @return
9  */
10 public Sqlresponse update(String tabla, List<String> campos, List<String>
11                           valores, List<List<Object>> ROWIDs) {
12     for (Object ite : ROWIDs) {
13
14         String peticion = "UPDATE " + tabla + " SET ";
15
16         for (int i = 0; i < campos.size(); i++) {
17             if (i > 0) {
18                 peticion = peticion + ",";
19             }
20             peticion = peticion + campos.get(i) + " = \\" + valores.get(i) +
21                         "\\";
22         }
23         peticion = peticion + " WHERE ROWID = " + "\\" + ((String) ((List<
24             String>) ite).get(0)) + "\\'; ";
25
26         Logger.getGlobal().fine(peticion);
27         this.sql(peticion);
28     }
29     return null;
30 }
```

Figura 12.9
Código: Métodos para Fusion Table: UPDATE

12.3.4 *Métodos de borrado: DELETE*

Esta función permite borrar una tupla concreta. Para ello, se requiere proporcionar el ROWID o identificador de tupla, que puede ser adquirido realizando una selección.

```

1  /**
2  * Función que elimina una tupla en la tabla alojada en FusionTables
3  *
4  * @param tabla identificador de la tabla
5  * @param ROWID identificador de la tupla
6  * @return
7  */
8  public Sqlresponse delete(String tabla, List<List<Object>> ROWIDs) {
9      String peticion;
10
11     for (Object ite : ROWIDs) {
12         peticion = "DELETE FROM " + tabla + "\ WHERE ROWID = " + "\'" + ((
13             String) ((List<String>) ite).get(0)) + "\';";
14         this.sql(peticion);
15     }
16
17     return null;
17 }
```

Figura 12.10
Código: Métodos para Fusion Table: DELETE

12.4 SISTEMAS DE ACTUALIZACIÓN

A lo largo de esta sección hemos presentado los métodos desarrollados para trabajar de forma nativa con operaciones SQL, pero aún no se ha mostrado el uso que hacemos de ellas en nuestro sistema.

En esta sección se muestran las submódulo de cálculo y subida a Fusion Tables de la información relativa a los nodos, los pasos y las trazas.

En los ejemplos que se muestran, se almacenan los valores agrupados por horas haciendo uso de los métodos presentados en el Capítulo 8.3.1.

12.4.1 Actualización de Nodos

Este subsistema se encarga de actualizar la información relativa a los nodos en una tabla de Google Fusion Tables. Para ello hace uso del método que presentamos a continuación:

```

1  public boolean calcular() {
2      Conectar conectar = new Conectar();
3      try {
4          Statement st = conectar.crearSt();
5          rs = st.executeQuery("select * from nodo");
6
7          List<String> valores = new ArrayList<>();
8
9          while (rs.next()) {
10              valores.add(rs.getString(1)); //idNodo
11              valores.add(rs.getString(2)); //latitud
12              valores.add(rs.getString(3)); //longitud
13              valores.add(rs.getString(4)); //nombre
14              valores.add(rs.getString(5)); //poligono
15
16              if(rs.getString("nombre").toLowerCase().contains("wifi")){
17                  valores.add("Wifi");
18              }else if(rs.getString("nombre").toLowerCase().contains("wi-fi")){
19                  valores.add("Wifi");
20              }else{
21                  valores.add("Bluetooth");
22              }
23              Logger.getGlobal().fine("Insertando valores del nodo"+valores.toString());
24              cFT.insert(TABLAID, campos, valores, check, 1);
25             /*Los nodos son especiales, dado que incorporan mucha más información, por
26              lo que es recomendables no usar el mecanismo de caché ya que se corre
27              el riesgo de desbordar el tamaño de la query. Es por ello, que forzamos
28              la sincronización después de cada insercción */
29              cFT.forzarSync();
30              Logger.getGlobal().fine("Nodo insertado");
31              valores.clear();
32          }
33          cFT.forzarSync();
34          cFT.esperarSubida();
35      } catch (SQLException ex) {
36          Logger.getGlobal().log(Level.SEVERE, null, ex);
37      }
38
39      return false;
40  }

```

Figura 12.11
Código: Fusion Tables: Subsistema de subida de nodos

12.4.2 Actualización de Pasos por Horas

Este subsistema se encarga de calcular los pasos que han transcurrido por cada nodo agrupados en intervalos y subirlos a una tabla de Google Fusion Tables. Hace uso de dos funciones principales. La primera de ella se encarga de localizar la fecha a la cual pertenecen los últimos temporales alojados en la tabla de Fusion Tables. Se presenta a continuación dicha función:

```
1 public String setFechaUltima() {
2     Sqlresponse r = cFT.select(TABLAID, "Intervalo", "Total>o", "ORDER BY `"
3         "Intervalo` DESC LIMIT 1");
4     String mifecha = (String) r.getRows().get(0).get(0);
5     try {
6         Date b = _d.sdf.parse((String) r.getRows().get(0).get(0));
7         Calendar c = Calendar.getInstance();
8         c.setTime(b);
9         c.add(Calendar.HOUR, -24);
10        mifecha = _d.sdf.format(b);
11        this.fecha = mifecha;
12        return fecha;
13    } catch (ParseException ex) {
14        Logger.getLogger(PasosPorHoras.class.getName()).log(Level.SEVERE, null,
15            ex);
16    }
17    //this.fecha = (String) r.getRows().get(0).get(0);
18    return mifecha;
19 }
```

Figura 12.12

Código: Fusion Tables: Subsistema de subida de Pasos por hora I

Como se puede observar, la ventana se amplía 24 horas. Esto es para tener la certeza de que todos los valores alojados en el sistema local se encuentran en el sistema en la nube.

A continuación la función que se encarga de calcular y subir los valores a Fusion Tables:

```

1 public boolean calcular() {
2     Conectar conectar = new Conectar();
3     try {
4         Statement st = conectar.crearSt();
5         Logger.getGlobal().log(Level.INFO, "Calculando pasos en DB LOCAL");
6         rs = st.executeQuery("CALL agrupaPasosPorIntervalosNodosSeparados('" +
7             fecha + "','" + _d.sdf.format(Calendar.getInstance().getTime()) +
8             "','" + 60 + "')");
9         List<String> valores = new ArrayList<>();
10        Logger.getGlobal().log(Level.INFO, "Subiendo información a la Nube");
11        while (rs.next()) {
12            valores.add(rs.getString(1)); //Intervalo
13            valores.add(rs.getString(2)); //idNodo
14            valores.add(rs.getString(3)); //Total
15            valores.add(rs.getString(7)); //poligono
16            try {
17                Logger.getGlobal().fine("Valores:" + rs.getString(2) + " " + rs.
18                    getInt(3) + " " + rs.getString(6));
19                infoNodo_pasoPorHora.update(rs.getString(2), rs.getInt(3), rs.
20                    getString(6));
21            } catch (Exception ex) {
22                Logger.getGlobal().log(Level.SEVERE, ex.getMessage());
23            }
24            cFT.insert(TABLAID, campos, valores, check, 2);
25            valores.clear();
26        }
27        Logger.getGlobal().log(Level.INFO, "Todos los valores procesados.");
28        cFT.forzarSync();
29        Logger.getGlobal().log(Level.INFO, "Esperando al envío y confirmación
30             de los valores en la nube.");
31        cFT.esperarSubida();
32        Logger.getGlobal().log(Level.INFO, "Todos los valores subidos a la nube
33             .");
34    } catch (SQLException ex) {
35        Logger.getGlobal().log(Level.SEVERE, "Fallo en cálculo de los pasos. " +
36            ex.getMessage(), ex);
37    }
38    return true;
39 }

```

Figura 12.13

Código: Fusion Tables: Subsistema de subida de Pasos por hora II

En dicho código se presenta la variable `infoNodo_pasoPorHora`. Dicha variable obedece a un sistema de cálculo de estadísticas en que se está trabajando para versiones futuras.

12.4.3 Actualización de Trazas por Horas

Al igual que el submódulo anterior, este módulo también calcula la última tupla insertada en la nube con una función similar, por lo que no será añadido el código perteneciente a dicha función. A continuación, el código de cálculo y subida de las trazas por horas:

```

1 public boolean calcular() {
2     Conectar conectar = new Conectar();
3     try {
4         Statement st = conectar.crearSt();
5         Logger.getGlobal().fine("CALL localizaTrazasNodos2('" + fecha + "','" + _d
6             .sdf.format(Calendar.getInstance().getTime()) + "','" + 60 + "')");
7         rs = st.executeQuery("CALL localizaTrazasNodos2('" + fecha + "','" + _d
8             .sdf.format(Calendar.getInstance().getTime()) + "','" + 60 + "')");
9         List<String> valores = new ArrayList<>();
10        while (rs.next()) {
11            //System.out.println("->" + rs.getString(1) + " " + rs.getString(2) + " "
12            //+ rs.getString(3));
13            valores.add(rs.getString(1)); //Fecha
14            valores.add(rs.getString(2)); //Origen
15            valores.add(rs.getString(3)); //Destino
16            valores.add(rs.getString(4)); //total
17            valores.add(rs.getString(5)); //Diferencia
18
19            String poligono = "<LineString> <coordinates> " + rs.getString(7) + ", "
20                + rs.getString(6) + ", " + rs.getString(9) + ", " + rs.getString(8) + ",
21                " + rs.getString(10) + " </coordinates> </LineString>";
22            valores.add(poligono); //nombre
23            double earthRadius = 3958.75;
24            double dLat = Math.toRadians(rs.getDouble(8)-rs.getDouble(6));
25            double dLng = Math.toRadians(rs.getDouble(9)-rs.getDouble(7));
26            double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
27                Math.cos(Math.toRadians(rs.getDouble(6))) * Math.cos(Math.toRadians(
28                    rs.getDouble(8))) *
29                Math.sin(dLng/2) * Math.sin(dLng/2);
30            double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
31            double dist = earthRadius * c;
32
33            int meterConversion = 1609;
34            float t = (float) dist * meterConversion;
35            valores.add(Float.toString(t));
36            cFT.insert(TABLAID, campos, valores, check);
37            valores.clear();
38        }
39        return false;
40    }

```

Figura 12.14
Código: Fusion Tables: Subsistema de subida de Trazas por hora

Para facilitar la representación, este método dibuja un círculo mediante KML para geoposicionar el nodo en un mapa de Fusion Tables. Esto fue cambiado durante una fase avanzada del proyecto, debido a que se decidió que la geolocalización fuese con cada paso en lugar de directamente en el nodo, debido a que los nodos podían ser cambiados de sitio, mostrando entonces el sistema una información errónea o no consistente.

12.5 CLIENTE ACTUALIZADOR FT

Todos estos submódulos son controlados por el módulo Actualizador FT que se presenta a continuación:

Este módulo, al igual que el actualizador local consta de dos métodos: uno para el primer arranque (donde todos los valores deben ser subidos por primera vez) y otro modo automático.

```

1 public void start() {
2     if (_c.getBool("ft.primera_vez")) {
3         Logger.getGlobal().log(Level.INFO, "Esperando 10 minutos para el
4             comienzo del sincronizado forzado en la NUBE");
5         try {
6             Thread.sleep(1000);
7         } catch (InterruptedException ex) {
8             Logger.getLogger(ActualizadorFT.class.getName()).log(Level.SEVERE,
9                 null, ex);
10        }
11        Logger.getGlobal().log(Level.INFO, "Comenzando sincronizado forzado en
12            la nube");
13
14        //Si es nuestra primera vez, habrá que hacerlo desde el origen de los
15        //tiempos!
16        Nodos n = new Nodos();
17        n.calcular();
18        Logger.getGlobal().log(Level.INFO, "Nodos sincronizados");
19        PasosPorHoras h = new PasosPorHoras("2014-07-01 00:00:00 ");
20        h.check = true;
21        h.calcular();
22        Logger.getGlobal().log(Level.INFO, "Pasos sincronizados");
23        TrazasPorHoras t = new TrazasPorHoras("2014-05-23 10:26:54 ");
24        t.check = true;
25        t.calcular();
26        Logger.getGlobal().log(Level.INFO, "Trazas sincronizadas");
27        _c.set("ft.primera_vez", "false");
28        Logger.getGlobal().log(Level.INFO, "Sincronizado forzado en la nube
29            programado COMPLETO.");
30    }
31    Timer timer = new Timer("ActualizadorFusionTable", true);
32    //timer.scheduleAtFixedRate(temporizador, _c.getLong("ft.
33    //periodo_actualizacion"), _c.getLong("ft.periodo_actualizacion"));
34    timer.scheduleAtFixedRate(temporizador, _c.getLong("ft.tiempo_espera"),
35        _c.getLong("ft.periodo_actualizacion"));
36}

```

Figura 12.15
Código: Fusion Tables: Cliente Actualizador de FT

Este módulo hace uso de un evento programado mediante el objeto temporizador que instanciado en el constructor tal y como muestra el siguiente código:

```

1     this.temporizador = new TimerTask() {
2         @Override
3         public void run() {
4             Logger.getGlobal().log(Level.INFO, "Comenzando sincronizado
5                 programado en la nube.");
6
7             if (conta_nodos == MAX_conta_nodos) {
8                 Nodos n = new Nodos();
9                 n.calcular();
10                Logger.getGlobal().log(Level.INFO, "Nodos sincronizados"
11                    );
12                conta_nodos = 0;
13
14            } else {
15                conta_nodos++;
16            }
17            infoNodo_pasoPorHora.reset();
18            PasosPorHoras h = new PasosPorHoras();
19            Logger.getGlobal().log(Level.INFO, "Sincronizando Pasos
20                desde " + h.getFecha());
21            h.check = true;
22            h.calcular();
23            Logger.getGlobal().log(Level.INFO, "Pasos sincronizados.");
24            Logger.getGlobal().log(Level.INFO, "Preparando envío de
25                Tweet.");
26            TwitterAgente _t = new TwitterAgente();
27            _t.publicar(infoNodo_pasoPorHora.prime());
28            TrazasPorHoras t = new TrazasPorHoras();
29            Logger.getGlobal().log(Level.INFO, "Pasos sincronizados
30                desde " + t.getFecha());
31            t.check = true;
32            t.calcular();
33            Logger.getGlobal().log(Level.INFO, "Trazas sincronizadas.");
34            Logger.getGlobal().log(Level.INFO, "Sincronizado programado
35                en la nube programado COMPLEJO.");
36            Estadisticas.prime();
37        }
38    };

```

Figura 12.16

Código: Fusion Tables: Cliente Actualizador de FT: Temporizador

Esta tarea es ejecutada cada X minutos por el módulo, en función de lo que indique el fichero de configuración.

Además, en este código se muestran dos módulos que aún no han sido presentados: el módulo de publicación twitter instanciado en la variable `_t` y el módulo de estadísticas en el que se está actualmente trabajando.

13

SIRVIENDO LOS DATOS: GOOGLE FUSION TABLES API REST

ÍNDICE DE CAPÍTULO

- 13.1 Referencia rápida de la API REST **160**
 - 13.1.1 Consultas sobre los datos: SELECT **160**
 - 13.1.2 Insercione sobre los datos: INSERT **162**
 - 13.1.3 Actualizaciones sobre los datos: UPDATE **163**
 - 13.1.4 Borrado de datos: DELETE **164**
 - 13.2 Ejemplo de uso **164**
 - 13.2.1 Impresión en tabla HTML **165**
 - 13.2.2 Mapa de Google Maps **166**
-

En este capítulo se define la API REST de comunicación de Google Fusion Tables [referencia a <https://developers.google.com/fusiontables/>] , recogiendo algunas de las peticiones más frecuentes así como la integración del servicio en otras plataformas de Google y el uso en cualquier Web mediante peticiones REST con Javascript.

13.1 REFERENCIA RÁPIDA DE LA API REST

13.1.1 Consultas sobre los datos: *SELECT*

Para realizar consultas sobre los datos, se realiza una petición HTTP GET al servidor de Google empleando la siguiente sintaxis:

```
"https://www.googleapis.com/fusiontables/v1/query?sql="
SELECT <column_spec> {, <column_spec>}*
FROM <table_id>
{ WHERE <filter_condition> | <spatial_condition> { AND <
    filter_condition> }* }
{ GROUP BY <column_name> {, <column_name>}* }
{ ORDER BY <column_spec> { ASC | DESC } | <
    spatial_relationship> }
{ OFFSET <number> }
{ LIMIT <number> }
```

Figura 13.1
Google Fusion Tables API: Select

Retorno

Si la consulta SELECT es procesada correctamente, se devuelve un JSON formateando los datos en UTF-8. Si se desea, se puede modificar la petición para recibir los datos en formato CSV.

Parámetros

<column_spec> Indica que columnas desean ser incluidas en la salida del SELECT. Cuando se usa con una sentencia ORDER BY

la columna por la que se ordena debe ser incluida en este conjunto. Posibles valores

<column_name> Descripto más adelante en este capítulo.

<aggregate_spec> Descripto más adelante en este capítulo.

rowid Identificador de tupla. Necesario para sentencias INSERT,UPDATE o DELETE.

* Operador comodín, permite devolver todas las columnas de la tabla.

<column_name> El nombre de la columna, es case sensitive.

<aggregate_spec> Función de agregación que especifica una función que será aplicada a todos los valores de una columna. Devuelve un valor numérico. No se puede emplear con la sentencia ORDER BY. Se puede emplear cualquier de las siguientes operaciones comunes a lenguajes SQL: COUNT, SUM, AVERAGE, MAXIMUM, MINIMUM.

<table_id> Identificador único de la tabla, a la cual el usuario tiene permiso de acceso.

<filter_condition> Expresión Booleana. Sólo las tuplas que cumplan la condición serán incluidas entre los datos devueltos. Se puede combinar con AND para unir varios filtros. El operador OR no está soportado.

<column_condition> Usado en las sentencias WHERE. Indica una condición que tiene que cumplir el valor de una tupla para ser incluida dentro de los datos devueltos. Se encuentran disponibles los siguientes operadores.

- Valores numéricos: <,>, >=, <=, =
- Cadenas de texto: <,>, >=, <=, =, LIKE, MATCHES, STARTS WITH, ENDS WITH, CONTAINS, CONTAINS IGNORING CASE, DOES NOT CONTAIN, NOT EQUAL TO, IN
- Columnas geolocalizadas: Están disponibles los mismo operadores que para **<spatial_condition>**.
- Columnas con fechas: Son tratadas como valores numéricos.
- NO SOPORTADO: Consultas sobre fórmulas o variables calculadas.

<row_condition> Indica una tupla concreta mediante una asignación ROWID = <string>.

<spatial_condition> Indica una zona concreta mediante una asignación ST_INTERSECTS (<location_column>, <geometry>).

Puede ser combinado con ORDER BY haciendo uso de la función ST_DISTANCE. Existen tres zonas <geometry> que pueden ser empleadas

- Círculos representados como CIRCLE(<coordinate>, <radius>).
- Rectángulos representados como RECTANGLE(<lower_left_corner>, <upper_right_corner>)
- Coordenadas representadas como LATLNG<nombre>, <number>. Advertencia: se expresan en orden contrario al estándar KML.

group by Agrupa los resultados de salida mediante un valor de una columna común. El propósito es calcular un resumen con columnas <aggregate_spec>.

order by Ordena la salida en función de una sola columna o una spatial_relationship representada como ORDER BY ST_DISTANCE(<, <coordinates>).

offset Omite los primeros <number> tuplas de la tabla.

limit Devuelve un máximo de <number> tuplas.

<number> Un número entero. No puede ser una expresión.

<location_columna> Una columna geoposicionada.

<geometry> Usar uno de los siguientes <circle> o <rectangle>.

<radius> Un entero que indica el radio de un círculo expresado en metros.

13.1.2 Insercción sobre los datos: INSERT

Para insertar una o más tuplas en la tabla, se usa la siguiente petición HTTP POST:

```
INSERT INTO <table_id> (<column_name> {, <column_name>}*)
VALUES (<value> {, <value>}*)

{ ;INSERT INTO <table_id> (<column_name> {, <column_name>}*)
VALUES (<value> {, <value>}*)}* ;}
```

Figura 13.2
Google Fusion Tables API: INSERT

Se pueden insertar un máximo de 500 tuplas o un tamaño máximo de petición de 1MB.

Retorno

Si tiene éxito la petición devuelve el ROWID asociado a la tupla recién insertada.

Parámetros

<table_id> Identificador único de la tabla a la cual el usuario tiene permisos de escritura.

<column_name> El nombre de una columna de la tabla. Es case sensitive.

<value> El número entero, en coma flotante o cadena de texto que quiere ser asignada a la columna. O una lista de dichos valores separados por comas. Para dejar un valor de columna sin asignar, se emplea el operador cadena vacía . En una columna con localización geográfica, se puede emplear una cadena de texto con la dirección postal, un par de columnas con la latitud/longitud o una cadena de texto que contenga un punto geográfico siguiendo el estándar KML siendo permitido especificar puntos, líneas o polígonos.

13.1.3 Actualizaciones sobre los datos: UPDATE

Para actualizar una o más columnas de una ÚNICA tupla, se usa la siguiente petición HTTP POST:

```
UPDATE <table_id>
SET <column_name> = <value> {, <column_name> = <value>}*
WHERE ROWID = <row_id>
```

Figura 13.3
Google Fusion Tables API: UPDATE

Retorno

Si la petición es procesada con éxito, no se devuelve ningún objeto, pero la petición devuelve un estado 200 OK.

Parámetros

<table_id> Identificador único de la tabla a la cual el usuario tiene permisos de escritura.

<column_namec> El nombre de una columna de la tabla. Es case sensitive.

<value> El número entero, en coma flotante o cadena de texto que quiere ser asignada a la columna. O una lista de dichos valores separados por comas. Para dejar un valor de columna sin asignar, se emplea el operador cadena vacía . En una columna con localización geográfica, se puede emplear una cadena de texto con la dirección postal, un par de columnas con la latitud/longitud o una cadena de texto que contenga un punto geográfico siguiendo el estándar KML siendo permitido especificar puntos, líneas o polígonos.

<row_id> El identificador único de tupla. Para conseguir dicho identificador se debe realizar una petición SELECT antes de realizar esta petición.

13.1.4 *Borrado de datos: DELETE*

Para borrar una tupla, se usa la siguiente petición HTTP POST:

```
DELETE FROM <table_id>{ WHERE ROWID = <row_id>}
```

Figura 13.4
Google Fusion Tables API: DELETE

Retorno

Si la petición es procesada con éxito, no se devuelve ningún objeto, pero la petición devuelve un estado 200 OK.

Parámetros

<table_id> Identificador único de la tabla a la cual el usuario tiene permisos de escritura.

<row_id> El identificador único de tupla. Para conseguir dicho identificador se debe realizar una petición SELECT antes de realizar esta petición.

13.2 EJEMPLO DE USO

En el siguiente apartado se resumen algunos ejemplos de uso de la API de Google Fusion Tables, así como de ejemplos de integración en distintos servicios de Google.

13.2.1 Impresión en tabla HTML

El siguiente código Javascript realiza una petición a la API REST para traer una estructura JSON con la información solicitada, la cual es procesada y representada en una tabla HTML

```

1 function drawTableNodos(contenedor, nombre) {
2     var query = "select col0>o, col3>>1, MAX(col1>>0) as 'Fecha
3         Implantación', MAX(MAX(col1>>0)) as 'Fecha Último Registro' from 1
4         WENRMKLPdI-8WCKEoP6PTCwRkqCn88tRg-WHfV group by col3>>1,col0>>o,
5         col5>>1";
6     var queryText = encodeURIComponent(query);
7     var gvizQuery = new google.visualization.Query(
8         'http://www.google.com/fusiontables/gvizdata?tq=' + queryText);
9
10    gvizQuery.send(function (response) {
11        console.log(response);
12        var numRows = response.getDataTable().getNumberOfRows();
13        var numCols = response.getDataTable().getNumberOfColumns();
14        var ftdata = ['<table class="table" style="overflow-x:scroll;" id="'
15            + nombre + '"><thead><tr>';
16        for (var i = 1; i < numCols; i++) {
17            var columnTitle = response.getDataTable().getColumnLabel(i);
18            ftdata.push('<th>' + columnTitle + '</th>');
19        }
20        ftdata.push('<th class="acciones">Acciones</th></tr></thead><tbody>'
21            );
22
23        for (var i = 0; i < numRows; i++) {
24            ftdata.push('<tr>');
25            var idNodo; var nameNodo;
26            for (var j = 0; j < numCols; j++) {
27                var rowValue = response.getDataTable().getValue(i, j);
28                if (j == 0) {
29                    idNodo = rowValue;
30                } else if (j != 1) {
31                    ftdata.push('<td>' + rowValue + '</td>');
32                } else {
33                    nameNodo = rowValue;
34                    ftdata.push('<td>' + rowValue + '</td>');
35                }
36            }
37            ftdata.push('<td class="center "><a class="btn btn-success"
38                onclick=\`showNodoInfo(' + idNodo + ',' + nameNodo + '\')\'
39                ><i class="icon-zoom-in icon-white"></i>Ver</a></td></tr>';
40        }
41        ftdata.push('</tbody></table>');
42        document.getElementById(contenedor).innerHTML = ftdata.join('');
43    });
44 }

```

Figura 13.5

Código: Ejemplo de uso de la API REST de Google Fusion Tables

13.2.2 Mapa de Google Maps

Google nos ofrece dos maneras de incorporar la información alojada en Google Fusion Tables en un mapa de Google Maps. La primera de ella es un mecanismo nativo, que aunque es bastante potente, ofrece poco mecanismos avanzados y de personalización. El segundo método presentado, hace uso de la API REST para realizar una petición y crear de forma manual la capa.

Forma nativa

Google Maps funciona mediante *capas* (layer), que no son más que gráficos geolocalizados que son superpuestos sobre el mapa. De forma nativa, Google permite crear una capa sobre un mapa. En el siguiente código, se muestra como añadir una capa con la información relativa a una capa de Fusion Tables mediante Javascript:

```
1 layer = new google.maps.FusionTablesLayer({      map: map,
2   heatmap: { enabled: false },
3   query: {
4     select: "poligono",
5     from: "1WENRMKLPPhLdl-8WCKEoP6PTCwRkqCn88RgWtHV",
6     where: "Intervalo = '2014-03-12 10:00:00'"
7   },
8   options: {
9     styleId: 3,
10    templateId: 5,
11    suppressInfoWindows: true
12  }
13});
```

Figura 13.6

Código: Ejemplo de uso de Google Fusion Tables en Google Maps.

Mediante API REST

Sin embargo, el método anterior no permite mucha personalización (aunque si lo suficiente para un uso básico) ni permite realizar modificaciones en los datos u asociarles funcionalidades extras. Es por ello, que se presenta el siguiente código donde se hace uso de una capa de Google Maps usando la API REST directamente. En este caso, para realizar una capa con un “mapa de calor”

```

1 var query = 'select latitud, longitud, Total from iWENRMKLPtLdl-8
2   WCKEoP6PTCwRkjCn88lRgWtHV where col1\x3e\x3eo \x3d \x27'+c+'\x27 limit
3     1000';var request = gapi.client.fusiontables.query.sqlGet({ sql: query
4   });
5
6
7 function onDataFetched(response) {
8   if (response.error) {
9     alert('Unable to fetch data. ' + response.error.message +
10       ' (' + response.error.code + ')');
11   } else {
12     drawHeatmap(extractLocations(response.rows));
13   }
14 }
15
16 function extractLocations(rows) {
17   var locations = [];
18   for (var i = 0; i < rows.length; ++i) {
19     var row = rows[i];
20     if (row[0]) {
21       var lat = row[0];
22       var lng = row[1];
23       if (lat && lng && !isNaN(lat) && !isNaN(lng)) {
24         var latLng = new google.maps.LatLng(lat, lng);
25         var weight = row[2];
26         locations.push({ location: latLng, weight: parseFloat(weight) });
27       }
28     }
29   }
30   console.log(locations);
31   return locations;
32 }
33
34 function drawHeatmap(locations) {
35   var heatmap = new google.maps.visualization.HeatmapLayer({
36     dissipating: true,
37     gradient: [
38       'rgba(102,255,0,0)',
39       'rgba(147,255,0,1)',
40       'rgba(193,255,0,1)',
41       'rgba(238,255,0,1)',
42       'rgba(244,227,0,1)',
43       'rgba(244,227,0,1)',
44       'rgba(249,198,0,1)',
45       'rgba(255,170,0,1)',
46       'rgba(255,113,0,1)',
47       'rgba(255,57,0,1)',
48       'rgba(255,0,0,1)'
49     ],
50     opacity: 0.8,
51     radius: 30,
52     maxIntensity: 1000,
53     data: locations
54   });
55   heatmap.setMap(map);
56 }

```

Figura 13.7

Código: Ejemplo de uso de la API REST de Google Fusion Tables en Google Maps.

14

DESARROLLO WEB Y MÓVIL

ÍNDICE DE CAPÍTULO

- 14.1 Servicios Web **170**
 - 14.1.1 Tablas **170**
 - 14.1.2 Gráficos **171**
 - 14.1.3 Mapas **172**
 - 14.2 Plataformas Web **177**
 - 14.2.1 Página principal: Wordpress **177**
 - 14.2.2 Visor de eventos online **179**
 - 14.2.3 Panel de control **180**
 - 14.2.4 Panel de Administración de Google Fusion Tables **183**
 - 14.3 Aplicación Móvil **184**
 - 14.3.1 Optimizaciones **184**
-

En los anteriores capítulos hemos mostrado el sistema de subida de datos a la Nube Goolge Fusion Tables así como la API REST que nos permite acceder a los datos desde aplicaciones de terceros. Adicionalmente a esta API, se han desarrollado varios servicios Web que permite la creación de mecanismos avanzados para la cración de tablas, gráficos y mapas embebidos.

Sobre estos servicios se ha desarrollado varias plataformas web que permite la difusión de los datos: una página principal de información general sobre el proyecto, un panel de visualización de los datos, un visor de eventos. Adicionalmente se presenta el panel de administración de los datos de Google Fusion Tables, que permite la gestión de los datos y su autoría.

Por último, se presenta la aplicación móvil desarrollada en Android usando tecnología HTML.

14.1 SERVICIOS WEB

Si bien disponemos de una API REST para el acceso y manipulación de los datos alojado en la nube, se ha desarrollado una capa de Servicios Web que hace uso de dicha API para crear elementos de representación de datos. Al hacerlo mediante una capa externa, en lugar de gestionarlo en la propia aplicación, se consigue un entorno uniforme y modular. Una actualización en un servicio web, implica que todas las demás plataformas y servicios que hacen uso de dicho servicio tengan al instante la mejora o nueva funcionalidad.

Además, se consigue consolidar una imagen y mecánicas de uso comunes a todos las plataformas de forma fácil y sencilla.

A continuación, se presentan los 3 tipos de visualización básicos empleados, mostrando para cada uno algunos ejemplos de código así como capturas de pantalla y explicaciones de la funcionalidad.

14.1.1 *Tablas*

La forma más básica de representar información es mediante una estructura tabular. Se provee de un servicio web básico capaz de procesar una consulta SQL en una tabla HTML. Parte del código de dicho servicio se puede encontrar en la Figura 13.5.

Se ha creado una capa Javascript por encima de la tabla HTML para dotarle de funcionalidad adicional no nativa, como la posibilidad de aplicar filtros, búsqueda y ordenación por columnas

Nombre	Fecha Implantación	Fecha Último Registro	Acciones
Sipesca 1 bluetooth Brenes	Tue Jan 28 2014 19:00:00 GMT+0100 (CET)	Sat May 31 2014 17:00:00 GMT+0200 (CEST)	Ver
Sipesca 1 wifi Brenes	Tue Jan 28 2014 19:00:00 GMT+0100 (CET)	Sat May 31 2014 17:00:00 GMT+0200 (CEST)	Ver
Sipesca 10 bluetooth A-8005	Thu Jan 30 2014 19:00:00 GMT+0100 (CET)	Sun Aug 10 2014 16:00:00 GMT+0200 (CEST)	Ver
Sipesca 10 wifi A-8005	Thu Jan 30 2014 20:00:00 GMT+0100 (CET)	Tue Sep 02 2014 11:00:00 GMT+0200 (CEST)	Ver
Sipesca 11 bluetooth Carmona	Mon Feb 03 2014 14:00:00 GMT+0100 (CET)	Fri Apr 04 2014 10:00:00 GMT+0200 (CEST)	Ver
Sipesca 11 wifi Carmona	Mon Feb 03 2014 14:00:00 GMT+0100 (CET)	Mon Mar 31 2014 12:00:00 GMT+0200 (CEST)	Ver
Sipesca 12 bluetooth La Rinconada	Wed Mar 05 2014 11:00:00 GMT+0100 (CET)	Tue Sep 02 2014 11:00:00 GMT+0200 (CEST)	Ver
Sipesca 12 wifi La Rinconada	Wed Mar 05 2014 11:00:00 GMT+0100 (CET)	Tue Sep 02 2014 11:00:00 GMT+0200 (CEST)	Ver
Sipesca 13 bluetooth San José	Wed Mar 05 2014 14:00:00 GMT+0100 (CET)	Tue Mar 18 2014 08:00:00 GMT+0100 (CET)	Ver
Sipesca 13 wifi San José	Wed Mar 05 2014 14:00:00 GMT+0100 (CET)	Mon May 12 2014 00:00:00 GMT+0200 (CEST)	Ver

Figura 14.1
Representación en modo de tabla de los datos

Mostrando 10 entradas

Filtrar:

Nombre

Fecha Implantación

Fecha Último Registro

Acciones

Sipesca 1 bluetooth Brenes

Tue Jan 28 2014 19:00:00
GMT+0100 (CET)

Sat May 31 2014 17:00:00
GMT+0200 (CEST)

Ver

Sipesca 1 wifi Brenes

Tue Jan 28 2014 19:00:00
GMT+0100 (CET)

Sat May 31 2014 17:00:00
GMT+0200 (CEST)

Ver

Sipesca 10 bluetooth A-8005

Thu Jan 30 2014 19:00:00
GMT+0100 (CET)

Sun Aug 10 2014 16:00:00
GMT+0200 (CEST)

Ver

Sipesca 10 wifi A-8005

Thu Jan 30 2014 20:00:00
GMT+0100 (CET)

Tue Sep 02 2014 11:00:00
GMT+0200 (CEST)

Ver

Sipesca 11 bluetooth Carmona

Mon Feb 03 2014 14:00:00
GMT+0100 (CET)

Fri Apr 04 2014 10:00:00
GMT+0200 (CEST)

Ver

Sipesca 11 wifi Carmona

Mon Feb 03 2014 14:00:00
GMT+0100 (CET)

Mon Mar 31 2014 12:00:00
GMT+0200 (CEST)

Ver

Sipesca 12 bluetooth La Rinconada

Wed Mar 05 2014 11:00:00
GMT+0100 (CET)

Tue Sep 02 2014 11:00:00
GMT+0200 (CEST)

Ver

Sipesca 12 wifi La Rinconada

Wed Mar 05 2014 11:00:00
GMT+0100 (CET)

Tue Sep 02 2014 11:00:00
GMT+0200 (CEST)

Ver

Sipesca 13 bluetooth San José

Wed Mar 05 2014 14:00:00
GMT+0100 (CET)

Tue Mar 18 2014 08:00:00
GMT+0100 (CET)

Ver

Sipesca 13 wifi San José

Wed Mar 05 2014 14:00:00
GMT+0100 (CET)

Mon May 12 2014 00:00:00
GMT+0200 (CEST)

Ver

Mostrando 1 a 10 de 37 entradas

Anterior

Siguiente

Se hace necesario destacar que la petición de información y procesamiento de la tabla es realizado en el lado del cliente, al estar realizada enteramente en Javascript.

14.1.2 Gráficos

Si bien una tabla puede servir para el análisis de los datos, estos adquieren mayor vistosidad en un gráfico. Debido a la facilidad de integración de Google Fusion Tables con Google Chart, es posible realizar gráficos de forma rápida y muy vistosa.

Se proveen dos tipos de gráficos de ejemplo, uno basado en tecnología CANVAS y otro basado en tecnología FLASH a modo de ejemplo.

Histórico: serie temporal

Este tipo de gráfico permite mostrar una serie temporal de forma interactiva, donde el usuario puede seleccionar. Consta de dos zonas, la inferior muestra toda la serie así como una ventana temporal, que permite hacer zoom en la zona principal en la serie.

Figura 14.2
Gráfico de serie temporal

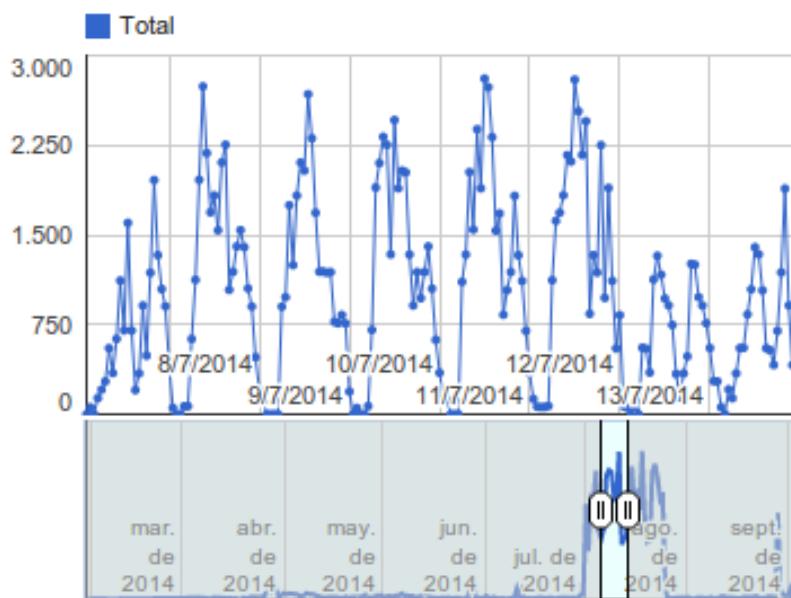


Gráfico interativo

Este tipo de gráfico sirven para contrastar la información de distintas columnas de la tabla, así como ofrecer animaciones de las vistas de los datos que pueden reproducirse (como una película) para mostrar la evolución a lo largo del tiempo.

Se puede elegir la columna representada por cada eje de la gráfica (indicando además si sigue un comportamiento lineal o logarítmico). Se pueden configurar los colores y tamaños de las representaciones para que sean automáticos, distinto para cada “tupla” o bien se correspondan con una escala de color basada en alguna columna.

Se disponen de tres gráficos a seleccionar: gráfico de dispersión XY, gráfico de barras y serie temporal. Las configuraciones realizadas en cada entorno son compartidas, lo que permite cambiar entre un tipo de visualización a otro de forma inmediata.

Sin embargo, este tipo de gráfica está limitada a entornos de escritorio con soporte a tecnología FLASH, por lo que no es soportado por dispositivos móviles.

14.1.3 Mapas

Este mapa permite navegar entre las fechas y horas

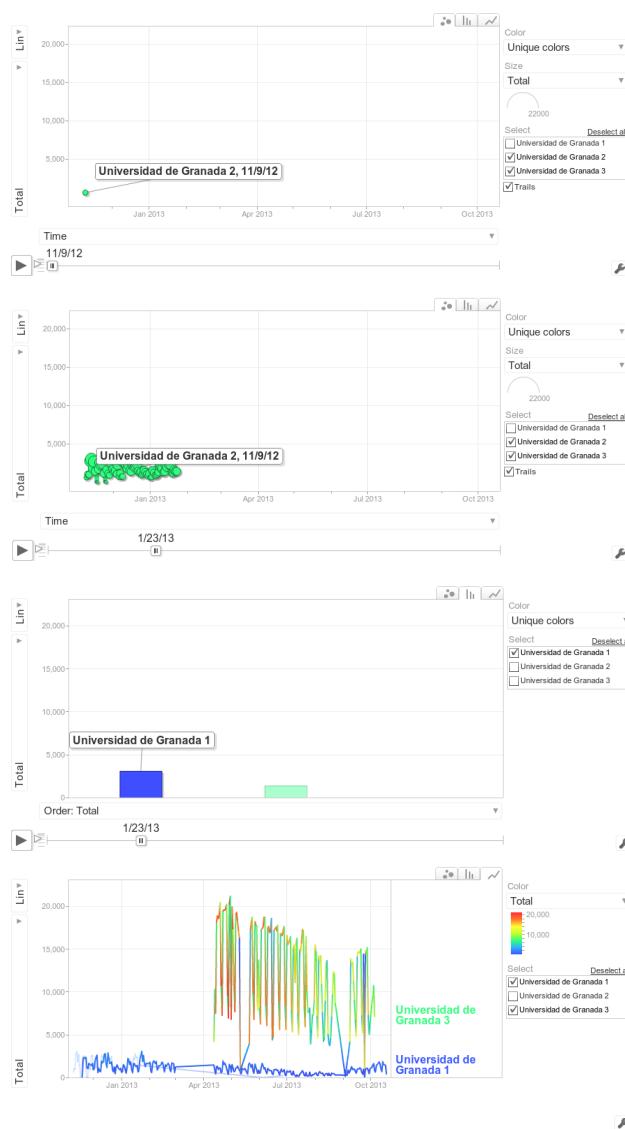


Figura 14.3
Animación de la
gráfica interactiva

Ya hemos visto con anterioridad lo fácil que resulta integrar un mapa en Google Maps con una fuente de datos basada en Google Fusion Tables.

Se monta un entorno basado en Google Maps en el que se añade funcionalidad extra para la visualización y manipulación de los datos:

Sobre dicho mapa se distinguen 4 zonas, que permiten manipular y navegar por los datos a visualizar:

Figura 14.4 Mapa principal de visualización de los datos

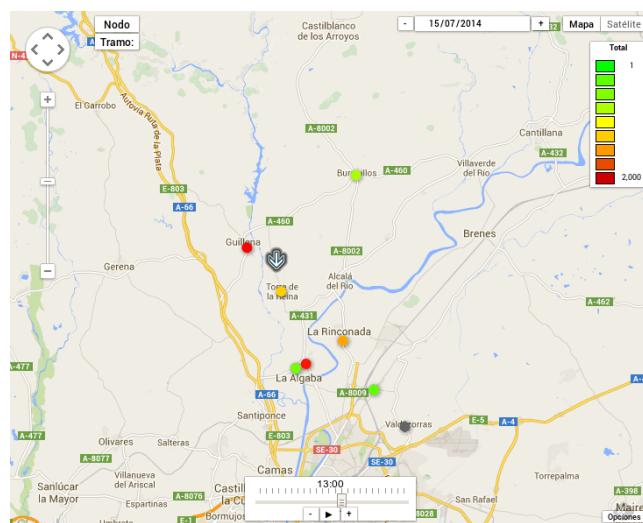
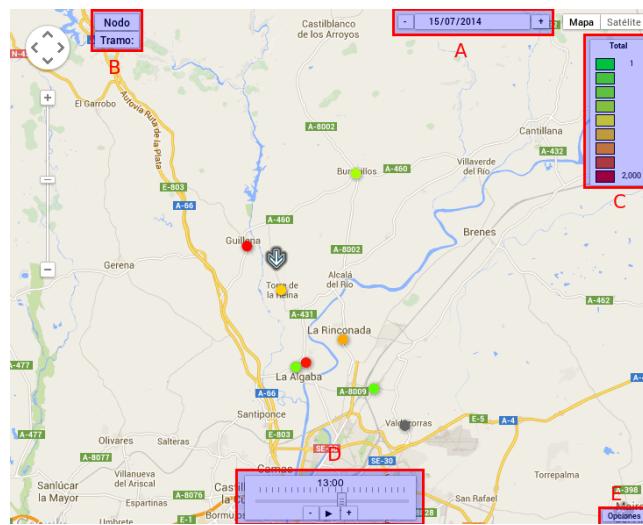


Figura 14.5
Mapa principal de visualización de los datos: Zonas de control



A: Zona de selección de fecha

Permite ajustar la fecha para la visualización de los datos. Por defecto muestra la fecha actual.

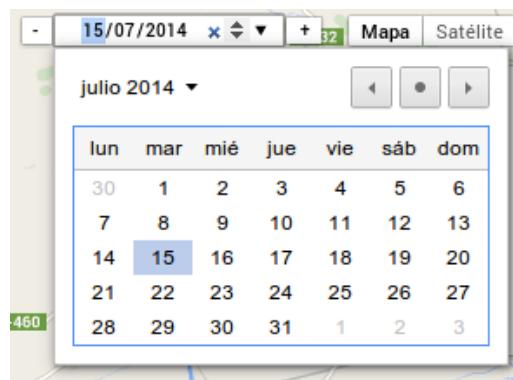


Figura 14.6
Mapa principal de visualización de los datos: Selector de fecha

B: Zona de Información Adicional

Cuando un nodo es seleccionado, muestra la información relativa a dicho nodo, así como ofrece la posibilidad de mostrar los gráficos relativos a la serie temporal de dicho nodo.



Figura 14.7
Mapa principal de visualización de los datos: Información del nodo

C: Leyenda de colores

Muestra la equivalencia en dispositivos detectados de la escala de colores empleada en la visualización de los datos.

D: Selector de hora

El selector de hora permite elegir la hora de la visualización de los datos. Los botones [1] y [3] permiten decrementar e incrementar la hora respectivamente, mientras que el botón [2] activa el modo de reproducción automática. El selector vertical [4] permite también ajustar de forma más rápida la hora que se desea visualizar.

Por defecto, muestra la hora actual.

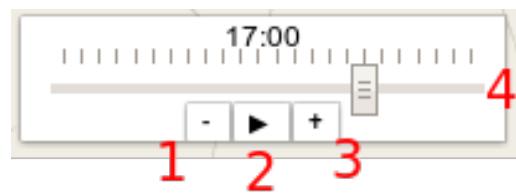


Figura 14.8
Mapa principal de visualización de los datos: Selector de hora

E: Opciones de configuración

Permiten ajustar la velocidad de las animaciones, el tipo de sensor que se muestra, así como permitir el “viaje rápido” entre las zonas en las que se encuentran los nodos.



Figura 14.9
Mapa principal de visualización de los datos: Opciones

14.2 PLATAFORMAS WEB

Una vez provistos de los mecanismos de representación de datos presentados anteriormente, se muestran las Plataformas Web desplegadas para el proyecto.

Cabe destacar en todos ellos un aspecto común, y es el empleo de una estructura Responsive Design, la cual permite ajustar de forma dinámica los contenidos visuales para ajustarlo al tamaño del dispositivo de forma fluida.

De esta forma, con una misma implementación se permite dar cabida a todos los entornos existentes.

14.2.1 *Página principal: Wordpress*

La página principal del proyecto, basada en una instalación de Wordpress en el servidor configurado en el Capítulo 9. Esta página ha sido usada como Landing Page del proyecto, ofreciendo la información tanto legal como del avance del proyecto. Además, ha servido de punto de contacto con los potenciales usuarios.

Dispone de la información relativa al proyecto, así como enlaces a los servicios de datos que serán presentados a lo largo de este capítulo.

Adicionalmente en esta página se recogen algunas de las series sociales en las que se ha tenido presencia del proyecto, como se tratará en el Capítulo 15.

El uso del software Wordpress para la realización de esta página de presentación se justifica por su rapidez de implantación, su flexibilidad y por la familiaridad con el entorno por parte de los investigadores involucrados en el proyecto.

Además, debido a ser un aspecto tangencial del proyecto, no resultaba fructífero invertir esfuerzos en realizar una implementación propia en este aspecto, además de que se hacía necesario disponer de esta información en la Web cuanto antes.

Más adelante, para el desarrollo del panel de control, si se realizaría una implementación propia que permitiese realizar ajustes correspondientes.

Figura 14.10
Web principal:
Vista de escritorio

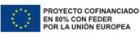
Figura 14.11
Web principal: Vis-
ta tablet y móvil

Tiempo	Etiqueta	Mensaje	Clase	Tarea
2014-09-01 18:47:09	INFO	Me voy a dormir 9.6686 minutos	ActualizadorLocal.ActualizadorDBLocal	run
2014-09-01 18:47:09	INFO	Podría actualizar cada 0.4142 minutos	ActualizadorLocal.ActualizadorDBLocal	run
2014-09-01 18:47:09	INFO	Me tengo que actualizar cada 10 minutos	ActualizadorLocal.ActualizadorDBLocal	run
2014-09-01 18:47:09	INFO	Última actualización hace 0.3314 minutos	ActualizadorLocal.ActualizadorDBLocal	run
2014-09-01 18:47:09	INFO	Estadísticas: Pasos Insertados: 0	Entorno.Estadisticas.Estadisticas	prime
2014-09-01 18:47:09	INFO	Estadísticas: Pasos Procesados: 0	Entorno.Estadisticas.Estadisticas	prime
2014-09-01 18:47:09	INFO	Estadísticas: Dispositivos Insertados: 0	Entorno.Estadisticas.Estadisticas	prime
2014-09-01 18:47:09	INFO	Estadísticas: Dispositivos Procesados: 0	Entorno.Estadisticas.Estadisticas	prime
2014-09-01 18:47:09	INFO	Estadísticas: Peticiones a DB generadas: 0	Entorno.Estadisticas.Estadisticas	prime
2014-09-01 18:47:09	INFO	Estadísticas: Peticiones a FT repetidas: 0	Entorno.Estadisticas.Estadisticas	prime

10 logs displayed, no new log found in [/var/log/apache2/error.log](#) with 3881 logs, 0 skipped line(s), 0 unreadable line(s).
File [/home/sipesca/sipesca.servidor/SiMa/logs/info.0](#) was last modified on 2014/09/01 18:47:09 at Europe/Madrid size is 10M

Este proyecto se está desarrollando gracias a la cofinanciación con fondos FEDER de la Unión Europea y por la Agencia de Obra Pública de la Consejería de Fomento y Vivienda de la Junta de Andalucía. Asimismo, queremos mostrar nuestro agradecimiento al personal e investigadores de la Agencia de Obra Pública de la Junta de Andalucía, Consejería de Fomento y Vivienda, por su dedicación y profesionalidad.

 Agencia de Obra Pública de la Junta de Andalucía
CONSEJERÍA DE FOMENTO Y VIVIENDA


 (evince:21212): C
 L (model) failed
 (evince:21212): C
 L (model) failed
[/var/log/apache2/error.log](#)

14.2.2 Visor de eventos online

Ya hemos hablado de la necesidad de depuración especiales que impone un sistema como el presentado en esta memoria en el Capítulo 7 donde se presentaba el sistema de depuración del sistema.

Sin embargo, dicho sistema se basa en la salida en ficheros formateados con la información de depuración lo cual resulta poco atractivo y funcional.

Para mejorar la depuración y control de errores del proyecto, se desarrolla un portal web que permite la visualización en tiempo real de la información de depuración del sistema.

Este sistema está basado en el software de visualización de logs del Sistema Apache aplicado a nuestro sistema SiMa.

Dispone de un sistema de “notificaciones” usando la api `web.notifications` que permite enviar notificaciones al sistema operativo mediante una página en el navegador web que cumpla el estándar.

Su desarrollo y uso ha resultado de vital importancia en la tarea de desarrollo y depuración de errores de todo el sistema.

Figura 14.12
Visor de eventos

14.2.3 *Panel de control*

De forma adicional a la página web (cuya finalidad es más de publicidad) y al visor de eventos (que obedece más a la necesidad de control del sistema) se presenta el panel de control desarrollado para la visualización de los datos públicos del sistema. Es por tanto, el sistema que emplearían los usuarios que quisieran obtener información sobre el sistema.

Al igual que las anteriores plataformas se haya implementado realizando un diseño Resonive Desing, para lo cual se ha empleado la conocida y popular librería *Bootstrap*.

Si bien el portal actual es sólo una maqueta de diseño, consta de funcionalidad completa en todos sus apartados y menús, debido principalmente al trabajo desarrollado en la implementación de los Servicios Web.

Debido a la normativa de la Ley de Cookies regulada por la Ley de Servicios de la Información, se ha implementado un servicio de doble almacenamiento de información, implementando un sistema basado tanto en LocalStorage como en Coockies. Esto nos permite emebeber nuestra portal Web como aplicación móvil, como veremos más adelante en la Sección [14.3](#).

Visita Guiada

Una funcionalidad a destacar es la funcionalidad que permite realizar una visita guiada sobre el portal web, que sirve tanto de tutorial como de presentación al entorno. Dicho tutorial se basa en la realización guiada de una serie de acciones y pasos sobre el portal, que permiten al usuario familiarizarse con el entorno.

Figura 14.13
Panel de control:
Visualización diversos dispositivos:
Escritorio, Tablet y Móvil

Figura 14.14

Distintos temas disponibles para el panel de control

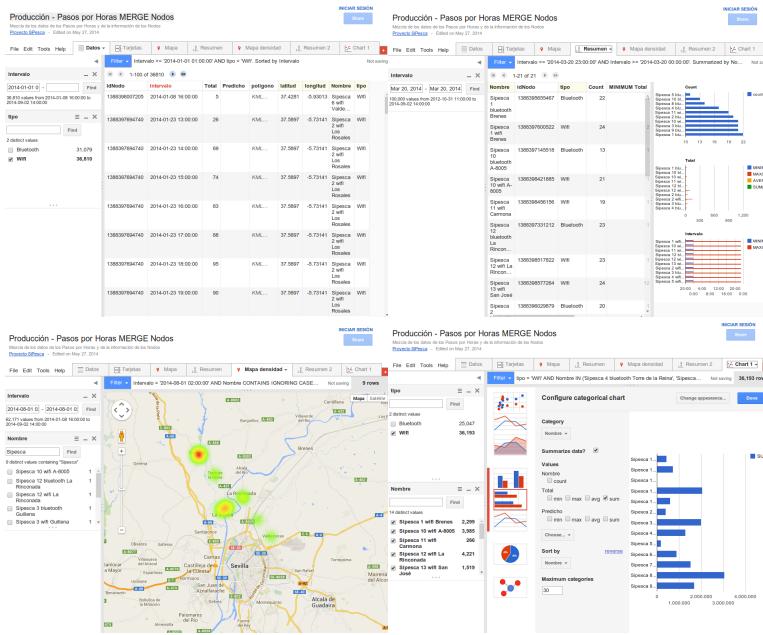


Figura 14.15
Panel de Administración de Google Fusion Tables:
Vista de tablas, resumen de datos,
Mapas de calor, Creación Interactiva
de Gráficas

14.2.4 Panel de Administración de Google Fusion Tables

Aunque no es fruto del desarrollo del Sistema, consideramos necesario hablar aunque sea brevemente del panel de control de Google Fusion Tables. Este entorno nos ofrece una interfaz web que nos permite trabajar con nuestros datos, así como representarlos en gráficos y mapas, mediante una interfaz interactiva.

Si bien no es posible realizar todas las funcionalidades avanzadas que permite la implementación con el uso de la API REST, es un buen entorno de visualización de los datos, pues permite de forma muy simple realizar operaciones de búsqueda y filtrado.

14.3 APPLICACIÓN MÓVIL

Aunque no estaba en la planificación inicial del proyecto, debido al auge de las tecnologías móviles y a la facilidad relativa para empaquetar todo nuestro entorno web en un app, se decidió crear una aplicación móvil para ofrecer información del servidor a los usuarios.

Para ello se realizó una aplicación basada en la tecnología de *Apache Cordova* y *Adobe Phonegap*, que componen un entorno libre y opensource para crear aplicaciones móviles basadas en estándares web.

A pesar de que este entorno permite el desarrollo para multitud de dispositivos móviles (iOS, Android, Windows Phone, Blackberry, FirefoxOS, webOS, Bada,...) usando el mismo componente web, a término de la redacción de este documento ha sido realizado la aplicación en Android.

La elección de este sistema operativo frente a otros, es su popularidad y facilidades que ofrece para el desarrollo de apps.

14.3.1 *Optimizaciones*

Sin embargo las aplicaciones basadas en estándares web difícilmente pueden igualar en rendimiento a las aplicaciones desarrolladas de forma nativa para una plataforma en concreto. Es por ello que las primeras versiones desarrolladas de nuestra app no hacía gala de un rendimiento cercano al fluido.

Se realizaron optimizaciones relativas tanto al entorno web como al entorno de la app. Por recoger en este documento sólo algunas de ellas:

- Habilitado del renderizado mediante hardware gráfico de la `WebView`.
- Inclusión de los recursos empleados por la app dentro de los `resources` en lugar de su descarga en cada ejecución.
- Optimizado del código Javascript encargado de la lógica de la app y unificado de los ficheros para reducir tiempos de carga.

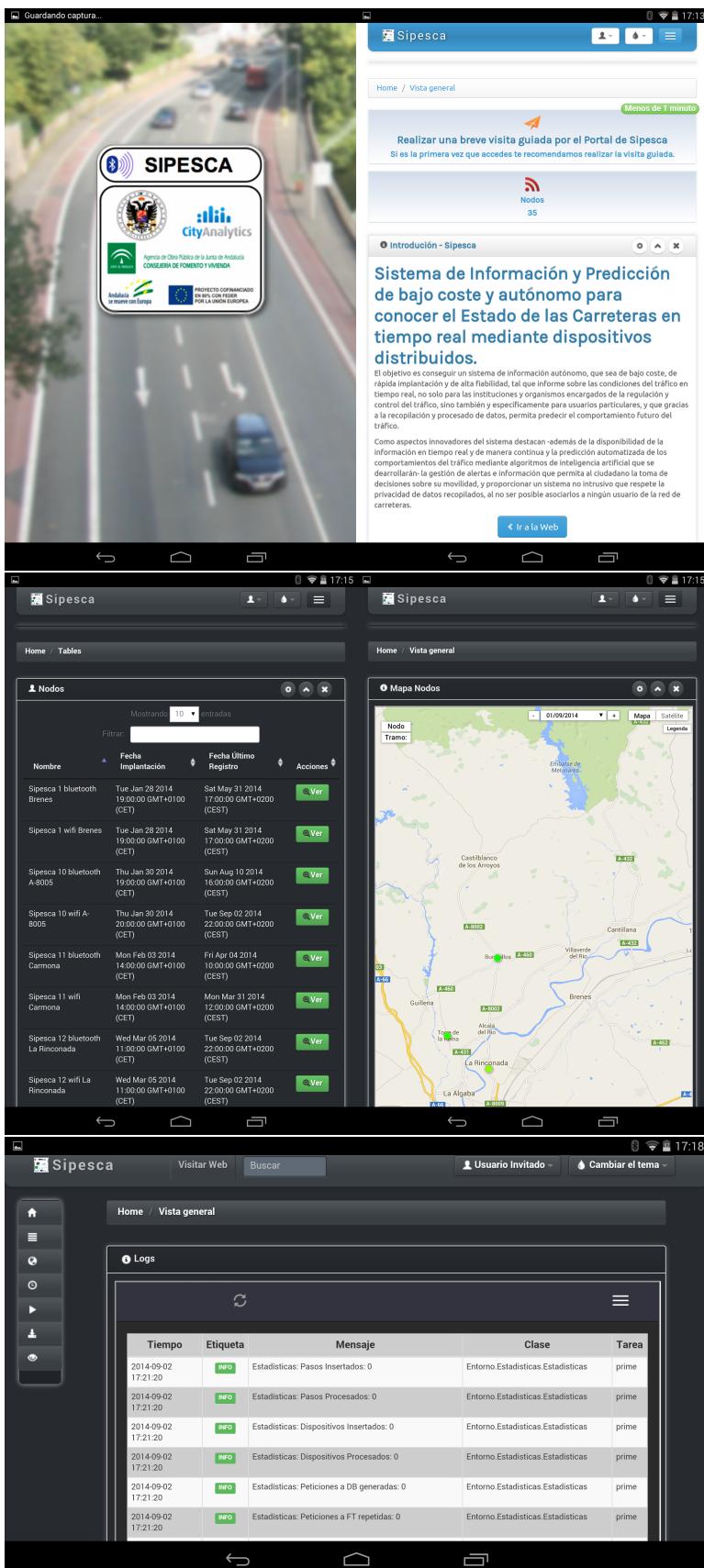


Figura 14.16
Capturas de pantalla de la aplicación móvil Android basada en tecnología Web

15

REDES SOCIALES, CIENCIA ABIERTA Y DIFUSIÓN

ÍNDICE DE CAPÍTULO

- 15.1 Redes Sociales **188**
 - 15.1.1 Twitter **188**
 - 15.1.2 Facebook **190**
 - 15.1.3 Youtube y Google Plus **191**
 - 15.2 Ciencia Abierta **192**
 - 15.2.1 Github **192**
 - 15.3 Difusión en prensa del proyecto **192**
-

La existencia de este breve capítulo en la redacción de esta memoria obedece al desempeño realizado en la difusión de la información relativa al proyecto SIEPEsCA en las redes sociales. Si bien parte de la experiencia de publicación en redes sociales ha sido llevada de forma personal por los investigadores del proyecto, en últimas fases de desarrollo del proyecto se realizó la implementación de un agente (o bot) de publicación Twitter.

Este agente permite la publicación en una cuenta de twitter especialmente dedicada a este cometido de la información en tiempo real del estado del tráfico. Sin embargo, la versatilidad y generalidad con la que ha sido diseñado permiten hacer uso de este módulo para la publicación periódica en Twitter de cualquier tipo de contenido que se deseé.

Además, se presenta brevemente el concepto de ciencia abierta bajo la cual se ha desarrollado todos los métodos, técnicas y sistemas presentados en este documento.

Por último, se recoge un breve resumen de las apariciones y menciones en prensa relativas al proyecto SIEPEsCA.

15.1 REDES SOCIALES

Se estima que más del 80 % de los usuarios de internet en España hace uso activo de las redes sociales [31], siendo las redes sociales más empleadas Facebook, Tuenti, Twitter y Youtube [9]. Es por ello que desde las fases más tempranas de desarrollo del proyecto se contempló la posibilidad de integración con dichas redes sociales¹

Si bien sólo se ha realizado un agente para la publicación de forma automática en Twitter, existen mecanismos propios a las redes sociales para compartir la información publicada entre ellas.

15.1.1 Twitter

Se crean dos cuentas de usuario de twitter, la primera de ellas la cuenta @sipesca con la información del avance del proyecto así como la redifusión de contenidos relacionados con el tráfico. Se crea adicionalmente la cuenta *traficosipesca* donde se publica de forma regular la información relativa al estado del tráfico en alguna de las zonas donde se encuentran ubicados los nodos.

¹ Salvo tuenti, que no fue considerado por tener un público más juvenil e infantil, no potencialmente interesado en el estado del tráfico.

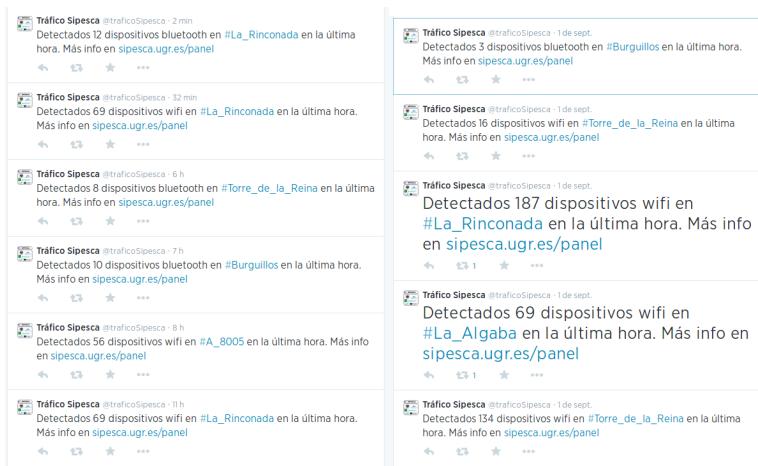


Figura 15.1
Ejemplos de tweets publicados de forma regular y automática por el sistema

Estos tweets hacen uso de un hashtag indicando la zona, por si el usuario desea suscribirse a la información de lugar concreto. Además, se ha pensado en la utilización de twitter para la notificación de otro tipo de eventos, siendo el módulo presentando a continuación solo un ejemplo de uso.

SiMa: Agente de Twitter

El agente de twitter implementado se basa en una implementación usando la librería de desarrollo de PHP de Twitter [22] a la cual se le envían peticiones mediante una URL local y privada al servidor desde un módulo de SiMa

```

1 <?php
2 require_once 'lib/twitter.class.php';
3 require_once 'lib/config.php';
4 $twitter = new Twitter($consumerKey, $consumerSecret, $accessToken,
5   $accessTokenSecret);
6 $m = substr($_GET['m'],0,140);
7 $m = str_replace("-", "_", $m);
8 try {
9   $tweet = $twitter->send($m);
10 }catch (TwitterException $e) {
11   echo 'Error: ' . $e->getMessage();
12 ?>

```

Figura 15.2
Código: Publicación en Twitter: Servicio privado PHP

```

1 public class TwitterAgente {
2     Config _c = new Config(); Debug _d = new Debug();
3
4     WebResource _w; Client client;
5
6     public TwitterAgente() {
7     }
8
9     public void publicar(String m){
10         if(!m.equals("")){
11             com.sun.jersey.api.client.config.ClientConfig config = new com.sun.
12                 jersey.api.client.config.DefaultClientConfig(); // SSL
13                 configuration
14             client = Client.create(config);
15             Logger.getGlobal().info("Voy a mandar un tweet: "+m);
16             _w = client.resource( _c.get("twitter.url") + "?m=" + m.replaceAll(" " ,
17                 "%20"));
18         }
19     }
20 }
```

Figura 15.3
Código: Publicación en Twitter: Java

Actualmente la información que se publica está relacionada con la cantidad de dispositivos detectados en un nodo concreto. Para ello se hace uso del módulo de estadísticas que está actualmente en desarrollo. Sólo es necesario conocer que en el proceso de subida de la información a la nube mediante Google Fusion Tables se elige un nodo concreto de forma pseudo-aleatoria y se muestran los dispositivos que se han detectado:

```

1 TwitterAgente _t = new TwitterAgente();
2
3 _t.publicar(infoNodo_pasoPorHora.prime());
```

Figura 15.4
Código: Publicación en Twitter: Java

Si se desea conocer más sobre este sistema, se invita a ver el código del mismo publicado según veremos en la Sección 15.2.1.

15.1.2 Facebook

Se ha querido tener presencia en Facebook replicando la información publicada mediante la cuenta de twitter @sipesca de forma automática.

Durante un primer momento, se replicó también la información publicada por la cuenta @traficosipesca, sin embargo debido al tráfico generado constantemente y a las quejas recibidas se desestimó.

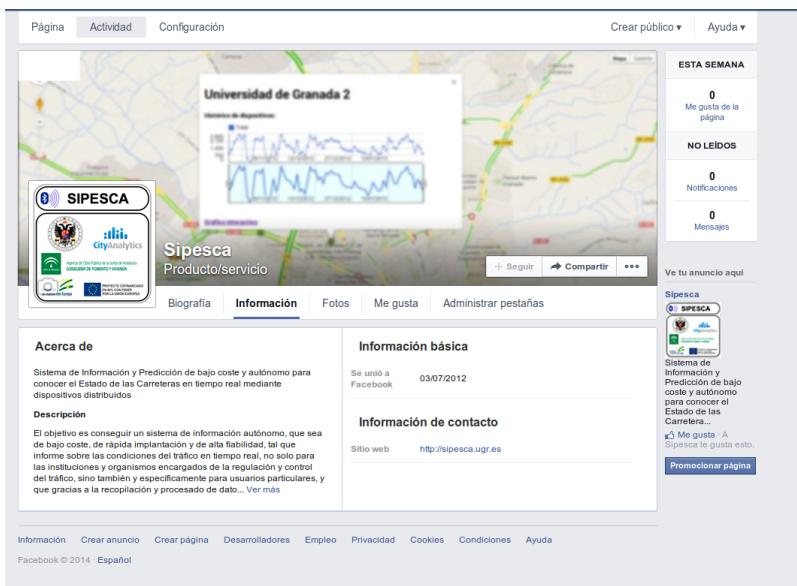
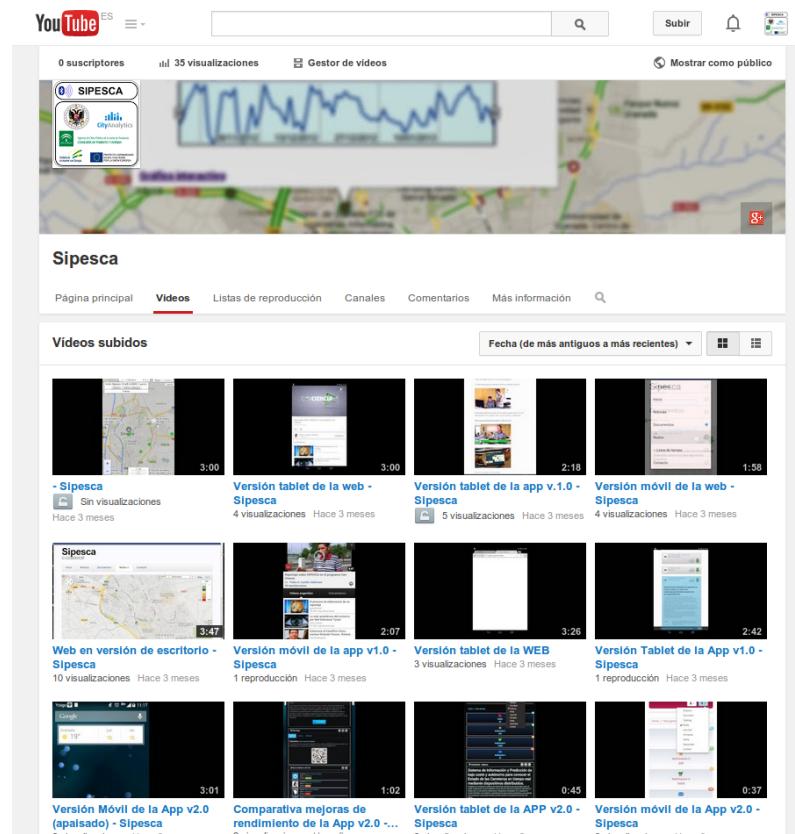


Figura 15.5
Página de Facebook del proyecto SIPESCA

15.1.3 YouTube y Google Plus

Debido a ser de los servicios de publicación vídeos más popular en Internet, se decidió abrir una cuenta para poder publicar los vídeos de ejemplo de las plataformas web y móviles desarrollados.

Figura 15.6
Página de YouTube del proyecto SipeSCA



15.2 CIENCIA ABIERTA

Todo el trabajo de implementación presentado en este proyecto se encuentra liberado bajo licencia GNU, por lo cual se permite la libertad de usar, estudiar, compartir y modificar el software.

15.2.1 Github

El código se encuentra publicado en la forja de código GITHUB [8], donde se puede encontrar la última versión estable de cada uno de los módulos del sistema, de los servicios web, las plataformas y la aplicación móvil presentados.

15.3 DIFUSIÓN EN PRENSA DEL PROYECTO

A continuación se recogen algunos enlaces publicados en medios tanto digitales como tradicionales sobre el proyecto SipeSCA.

- http://secretariageneral.ugr.es/pages/tablon/*/noticias-canal-ugr/un-nuevo-sistema-informatico-economico-y-facil-de-implantar-permite-conocer-el-trafico-en-tiempo-real-y-con-mayor-precision#.U2i2CC-x22x
- <http://www.20minutos.es/noticia/2130778/0/>
- <http://www.ideal.es/miugr/noticias/nuevo-sistema-informatico-permite-201405061225.html>
- <http://www.radiogranada.es/2014/05/06/disenan-un-nuevo-sistema-para-conocer-el-trafico-en-tiempo-real/>
- <http://www.webgranada.com/noticias.asp?modo=ver&ref=27319>
- <http://www.europapress.es/andalucia/noticia-nuevo-sistema-informatico-permite-conocer-trafico-tiempo-real-mayor-precision-20140506122258.html>
- http://www.infocostatropical.com/noticia.asp?id=56601&id_area=1&area=portada&id_seccion=17&sccion=Provincia
- <http://www.dicyt.com/noticias/sistema-para-conocer-el-trafico-en-tiempo-real>
- <https://es.cars.yahoo.com/noticias-coches/un-nuevo-sistema-inform%C3%A1tico-permite-101200185--spt.html>
- <http://www.autopista.es/noticias-motor/articulo/nuevo-sistema-conocer-estado-trafico-universidad-granada-100482>
- OndaCeroRadio(noticiasmediodía)
- CanalSurRadio-HoraSurMediodía(13:10h)
- <http://alacarta.canalsur.es/rss/programa/hora-sur-mediodia/220>
- [http://canal.ugr.es/buscar?searchword=72690&ordering=newest&searchphrase=all&areas\[0\]=digitales&areas\[1\]=impresos](http://canal.ugr.es/buscar?searchword=72690&ordering=newest&searchphrase=all&areas[0]=digitales&areas[1]=impresos)
- <http://www.granadaenlared.com/2014/05/06/un-nuevo-sistema-informatico-economico-y-facil-de-implantar-permite-conocer-el-trafico-en-tiempo-real-y-con-mayor-precision/>
- <http://www.autocasion.com/actualidad/noticias/156276/un-nuevo-sistema-informatico-permite-conocer-el-trafico-en-tiempo-real/>
- http://noticias.lainformacion.com/economia-negocios-y-finanzas/tecnologia-inalambrica/un-nuevo-sistema-informatico-permite-conocer-el-trafico-en-tiempo-real-y-con-mayor-precision_uWeKsk2edn0w6b7rI9TVC1/

- <http://www.elboletin.com/hoy-en-la-red/97847/sistema-informatico-conocer-trafico.html>
- <http://elcorreoeweb.es/2014/05/06/el-trafico-en-tiempo-real/>
- <http://www.economiadehoy.com/noticias/tecnologia/65327-un-nuevo-sistema-informatico-economico-y-facil-de-implantar-permite-conocer-el-trafico-en-tiempo-real-y-con-mayor-precision.html>
- <http://www.teleprensa.es/granada/un-nuevo-sistema-informatico-economico-y-facil-de-implantar-permite-conocer-el-trafico-en-tiempo-real.html>
- <http://www.abc.es/motor-reportajes/20140508/abci-sistema-trafico-universidad-granada-201405071401.html>
- "Cosasquepasan"-CanalExtremaduraRadio
- <http://www.canalextremadura.es/node/95074minuto46:21a53:30>
- LaCope-ElprogramadeRamónGarcía(alas16h)
- <http://www.cope.es/Menu/Podcast/la-tarde/podcast-la-tarde>
- http://vod.cope.es/audio/2014/05/09/audio_13996505795270376387.mp3minuto15:15a19:42
- RadioGranadaInformación-LavozdeGranada(alas17h)
- <http://www.lavozdegranadaradio.es/>

Parte VI

RESULTADOS Y CONCLUSIONES

16

CONCLUSIONES

En el ámbito del proyecto SIPEsCA se está desarrollando un sistema de información de bajo coste y autónomo para monitorizar el tráfico y conocer el estado de las carreteras en tiempo real.

El objetivo final es disponer de información acerca de los flujos de tráfico que se producen entre ciudades, para facilitar la gestión de la red viaria y la toma de decisiones en los desplazamientos por parte de los ciudadanos.

Gracias al dispositivo autónomo Intelify, el sistema recopila información de forma masiva para su posterior interpretación buscando, por ejemplo, asociaciones dentro de los datos obtenidos, que identifiquen tendencias.

En una primera fase, se instalaron seis dispositivos para monitorizar el tráfico teniendo en cuenta diferentes tipos de vía, con muy distintos tipos de tráfico.

Así, pudimos monitorizar la densidad de tráfico y los desplazamientos realizados por los usuarios, individualizando los vehículos conforme se mueven entre nodos dentro de la zona.

También obtuvieron estadísticas en función de los días de la semana y de los horarios de circulación.

Se realizaron diversos análisis de los datos recopilados durante un periodo de monitorización de dos meses para obtener diferentes estadísticas. Concretamente, el número total de vehículos detectados por cada nodo, en días laborables o festivos. También la densidad del tráfico por rango horario, y los desplazamientos individuales. Además, analizó la velocidad media en un tramo delimitado por dos nodos consecutivos.

Posteriormente, se fue mejorando la estructura de almacenamiento de la base de datos, usando servicios en la nube de Google para asegurar la disponibilidad. Asimismo, se realizó un trabajo intenso para la optimización de las consultas, y se actualizó el hardware del servidor para soportar un mayor tráfico y carga computacional.

También se recopilaron y probaron diversos métodos de data mining y procesamiento de datos, concretamente, algoritmos de clustering, clasificación y predicción de series temporales.

En una segunda fase, y a partir de una mejora en el hardware del dispositivo de monitorización, se instalaron los nodos en el área de Sevilla con lo que completamos la red necesaria para recopilar y ofrecer una información más completa y disponer de nuevos datos para continuar el trabajo.

Se determinó la necesidad de mantener los datos precalculados en la nube (Google Fusion Tables), asegurando la fiabilidad y disponibilidad de la información para los usuarios.

Asimismo, se diseñó y desarrolló una aplicación móvil (basada en tecnología web) multiplataforma para facilitar el acceso a la información a cualquier usuario, y desde cualquier plataforma (PC, móvil, tablet, etc).

Así, a través de la web principal y de la aplicación móvil comenzamos a servir información extraída de los datos a los usuarios, que pueden acceder a la información desde ordenadores, móviles y tablets.

En la última fase del proyecto, se implementó la tecnología de alimentación basada en placas solares y baterías, así como la detección por RFID.

Se mejoró el portal web y la aplicación web con la que se sirve la información sobre tráfico monitorizado. Se amplió y completó con diferentes tipos de información, estadísticas y gráficos interactivos.

Esta información se ofrece también a través de una aplicación móvil para la plataforma Android (aunque se podría adaptar fácilmente para otras plataformas móviles, como iOS, FirefoxOS o Windows Phone).

Además, se comenzó a servir información a través de diversas redes sociales, a las que los usuarios se pueden suscribir para recibir la información de los nodos en los que están interesados.

FUTURAS LÍNEAS DE INVESTIGACIÓN

Si bien el proyecto SIPeSCA se encuentra actualmente finalizado, se plantea seguir trabajando en los siguientes puntos a corto plazo:

- Instalar más dispositivos de monitorización ampliando el área urbana cubierta, lo cual permita una malla de dispositivos mayor.
- Instalar dispositivos en distintas ciudades para analizar el flujo de vehículos entre estas.
- Comprobar la efectividad de nuevos métodos de predicción de series temporales para integrarlos en el sistema.
- Usar nuevas redes sociales y herramientas móviles para ofrecer la información a los usuarios.
- Ampliar la aplicación móvil para servir información de manera más personalizadas.

En cuanto a las líneas futuras, a medio o largo plazo, cabe destacar:

- Realizar más análisis “big data” de los datos (no centrándose sólo en pasos y trazos).
- Desarrollar aplicaciones móviles para nuevos sistemas (Windows Phone, iPhone/iPad, Android L, Android Wear, Google Glass, Firefox OS).
- Analizar los datos con nuevas herramientas avanzadas de data-mining.
- Migrar el servidor local a un servidor en la nube como MS Azure o Amazon EC2 para asegurar la disponibilidad.

Parte VII

ANEXOS



BLUETOOTH

ÍNDICE DE CAPÍTULO

A.1	Service Class: Clase de servicio	206
A.2	Clases de dispositivo	206
A.2.1	Major Device Class	206
A.2.2	Minor Device Classes	207
A.2.3	Toy Major Class	209

En este capítulo se describen los campos que identifican a un dispositivo bluetooth [3, 5].

Cada dispositivo Bluetooth incorpora en la cabecera de nivel de Banda Base (Baseband 1.1) de sus paquetes un campo Class of Device/Service. Este campo se compone de 3 octetos organizados con el siguiente formato (en little endian):

- 11 últimos bits reservados para ServiceClasses
- 11 siguientes bits reservados para DeviceClasses
 - 6 últimos bits reservados para MajorDeviceClasses
 - 5 siguientes bits reservados para MinorDeviceClasses
- 2 primeros bits para el campo Format Type, por defecto o

A.1 SERVICE CLASS: CLASE DE SERVICIO

El campo reservado para las Service Classes permite identificar los servicios soportados por el dispositivo. Este campo se compone de 11 bits, del 13 al 23. Cada servicio Bluetooth está asociado a un bit en concreto, de forma que si un determinado bit del campo está a 1, entonces el dispositivo soporta ese servicio Bluetooth. La correspondencia entre nº de bit y servicio se recoge en la siguiente tabla:

Bit	Major Service Class
13	Limited Discoverable Mode
14	(reserved)
15	(reserved)
16	Positioning (Location identification)
17	Networking (LAN, Ad hoc, ...)
18	Rendering (Printing, Speaker, ...)
19	Capturing (Scanner, Microphone, ...)
20	Object Transfer (v-Inbox, v-Folder, ...)
21	Audio (Speaker, Microphone, Headset service, ...)
22	Telephony (Cordless telephony, Modem, Headset service, ...)
23	Information (WEB-server, WAP-server, ...)

Tabla A.1
Anexo: Bluetooth - Clases de servicio

A.2 CLASES DE DISPOSITIVO

El campo reservado para las Device Classes permite identificar la naturaleza del dispositivo. Este campo se compone 2 subcampos: Major Device Classes y Minor Device Classes

A.2.1 *Major Device Class*

El campo reservado para las Major Device Classes permite identificar el tipo genérico de dispositivo. Este campo se compone de 5 bits, del 8 al 12. Cada tipo genérico de dispositivo está asociado a una representación concreta de bits dentro del campo. La correspondencia entre bits y tipos genéricos de dispositivos se recoge en la siguiente tabla: Citar

12	11	10	9	8	Major Device Class
0	0	0	0	0	Miscellaneous
0	0	0	0	1	Computer (desktop,notebook, PDA, organizers,)
0	0	0	1	0	Phone (cellular, cordless, payphone, modem, ...)
0	0	0	1	1	LAN /Network Access point
0	0	1	0	0	Audio/Video (headset,speaker,stereo, video display, ...)
0	0	1	0	1	Peripheral (mouse, joystick, keyboards, ...)
0	0	1	1	0	Imaging (printing, scanner, camera, display, ...)
0	0	1	1	1	Wearable (complemento que puedes llevar puesto)
0	1	0	0	0	Toy (Juguete)
1	1	1	1	1	Uncategorized, specific device code not specified
X	X	X	X	X	All other values reserved

Tabla A.2

Anexo: Bluetooth - Major Device Class

A.2.2 Minor Device Classes

El campo reservado para las Minor Device Classes permite identificar el tipo específico de dispositivo. Este campo se compone de 6 bits, del 7 al 2. Cada tipo específico de dispositivo está asociado a una representación concreta de bits dentro del campo. La correspondencia entre bits y tipos específicos de dispositivos, dentro de cada tipo genérico, se recoge en la siguientes tablas:

Phone Major Class

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Uncategorized, code for device not assigned
0	0	0	0	0	1	Cellular
0	0	0	0	1	0	Cordless
0	0	0	0	1	1	Smart phone
0	0	0	1	0	0	Wired modem or voice gateway
0	0	0	1	0	1	Common ISDN Access
X	X	X	X	X	X	All other values reserved

Tabla A.3

Anexo: Bluetooth - Phone Major Class

Audio/Video Major Class

7	6	5	4	3	2	Minor Device Class
0	0	0	0	0	0	Uncategorized, code for device not assigned
0	0	0	0	0	1	Wearable Headset Device
0	0	0	0	1	0	Hands-free Device
0	0	0	0	1	1	(Reserved)
0	0	0	1	0	0	Microphone
0	0	0	1	0	1	Loudspeaker
0	0	0	1	1	0	Headphones
0	0	0	1	1	1	Portable Audio
0	0	1	0	0	0	Car audio
0	0	1	0	0	1	Set-top box
0	0	1	0	1	0	HiFi Audio Device
0	0	1	0	1	1	VCR
0	0	1	1	0	0	Video Camera
0	0	1	1	0	1	Camcorder
0	0	1	1	1	0	Video Monitor
0	0	1	1	1	1	Video Display and Loudspeaker
0	1	0	0	0	0	Video Conferencing
0	1	0	0	0	1	(Reserved)
0	1	0	0	1	0	Gaming/Toy
X	X	X	X	X	X	All other values reserved

Tabla A.4
Anexo: Bluetooth - Audio/Video Major Class

Peripheral Major Class

7	6	Minor Device Class
0	0	Other (Joystick, Gamepad, Remote control, ...)
0	1	Keyboard
1	0	Pointing device
1	1	Combo keyboard/pointing device

Tabla A.5
Anexo: Bluetooth - Peripheral Major Class

Imaging Major Class

7	6	5	4	Minor Device Class		
X	X	X	1	Display		
X	X	1	X	Camera		
X	1	X	X	Scanner		
1	X	X	X	Printer		
X	X	X	X	All other values reserved		

Tabla A.6

Anexo: Bluetooth - Imaging Major Class

Wearable Major Class

7	6	5	4	3	2	Minor Device Class	
o	o	o	o	o	o	Uncategorized, code for device not assigned	
o	o	o	o	o	1	Wrist Watch	
o	o	o	o	1	0	Pager	
o	o	o	o	1	1	Jacket	
o	o	o	1	o	o	Helmet	
o	o	o	1	o	1	Glasses	
X	X	X	X	X	X	All other values reserved	

Tabla A.7

Anexo: Bluetooth - Wearable Major Class

Toy Major Class

7	6	5	4	3	2	Minor Device Class	
o	o	o	o	o	o	Uncategorized, code for device not assigned	
o	o	o	o	o	1	Robot	
o	o	o	o	1	0	Vehicle	
o	o	o	o	1	1	Doll / Action Figure	
o	o	o	1	o	o	Controller	
o	o	o	1	o	1	Game	
X	X	X	X	X	X	All other values reserved	

Tabla A.8

Anexo: Bluetooth - Toy Major Class

BIBLIOGRAFÍA

- [1] Documentación *mecanismos autorización google fusion tables*. <https://developers.google.com/fusiontables/docs/v1/using#auth>. (Cited on page 140.)
- [2] Documentación *oauth2*. <https://developers.google.com/accounts/docs/OAuth2>. (Cited on page 140.)
- [3] Ieee bt (2013). *the ieee public bt oui listing*. retrieved dec 30, 2013, from <http://standards.ieee.org/develop/regauth/oui/oui.txt>. (Cited on page 205.)
- [4] *mount(8) Linux Man Page*. <http://linux.die.net/man/8/mount>. (Cited on page 26.)
- [5] Wikipedia *oui* (2013). *organizationally unique identifier*. retrieved dec 30, 2013, from http://en.wikipedia.org/wiki/Organizationally_unique_identifier. (Cited on page 205.)
- [6] Fhwa *traffic monitoring guide tmg - traffic monitoring guide*. April 2008. (Cited on page 4.)
- [7] UIMA Apache: *Apache software foundation*. URL <http://java.apache.org>, 2011. (Cited on page 29.)
- [8] A. Fernández Ares, P. Castillo Valdivieso, and J. Asensio Montiel: *Repositorio de código del proyecto sipesca*, 2013. <https://github.com/Sipesca/>. (Cited on page 192.)
- [9] H. Azevedo and P. Sánchez Martínez: *2013 spain digital future in focus*, 2013. <http://www.comscore.com/esl/Insights/Presentations-and-Whitepapers/2013/2013-Spain-Digital-Future-in-Focus>. (Cited on page 188.)
- [10] I.A Basheer and M Hajmeer: *Artificial neural networks: fundamentals, computing, design, and application*. Journal of Microbiological Methods, Volume 43, Issue 1, 1 December 2000, Pages 3-31. (Cited on page 117.)
- [11] B. Bowerman, R. O'Connell, and A. Koehler: *Forecasting: methods and applications*. Thomson Brooks/Cole: Belmont, CA. 2004. (Cited on page 118.)

- [12] G.E. Box and G.M. Jenkins.: *Time series analysis: forecasting and control.* Ed: San Francisco: Holden Day, 1976. (Cited on page 115.)
- [13] P. Brockwell and R. Hyndman: *On continuous-time threshold autoregression.* International Journal of Forecasting 8 (2) (1992) 157-173. (Cited on page 115.)
- [14] P. A. Castillo, Juan Julián Merelo, A. Prieto, and G. Romero. V. Rivas: *G-prop: Global optimization of multilayer perceptrons using gas.* Neurocomputing 35 (1-4). pp. 149-163. 2000. (Cited on page 114.)
- [15] P.A. Castillo, J. Carpio, Juan Julián Merelo, V. Rivas, G. Romero, and A. Prieto.: *Evolving multilayer perceptrons. neural processing letters.* (Cited on page 114.)
- [16] K. Chan and H. Tong: *On estimating thresholds in autoregressive models.* Journal of Time Series Analysis 7 (3) (1986) 179-190. (Cited on page 115.)
- [17] M. Clements, P. Franses, and N. Swanson: *Forecasting economic and financial time-series with non-linear models.* International Journal of Forecasting 20 (2) (2004) 169-183. (Cited on page 115.)
- [18] E.Parras-Gutierrez, M.Garcia-Arenas, V.M.Rivas, , and M.J. del Jesus: *Coevolution of lags and rbfn for time series forecasting: L-co-r algorithm.* Soft Computing, 16 (6), 919-942. 2012. (Cited on pages 114 and 115.)
- [19] G.Box and G.Jenkins: *Time series analysis: forecasting and control.* San Francisco: HoldenDay, 1976. (Cited on pages 114 and 115.)
- [20] Hector Gonzalez, Alon Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen: *Google fusion tables: Data management, integration and collaboration in the cloud.* In *SoCC10 Proceedings of the 1st ACM symposium on Cloud computing*, pages 175–180. ACM, 2010. (Cited on page 102.)
- [21] Hector Gonzalez, Alon Y. Halevy, Christian S. Jensen, Anno Langen, and Warren Shen Jonathan Goldberg Kidon Jayant Madhavan, Rebecca Shapley: *Google fusion tables: Web-centered data management and collaboration.* In *SIGMOD10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1061–1066. ACM, 2010. (Cited on page 102.)

- [22] Davud Grudl: *Twitter for php*, 2008. <http://dev.twitter.com/doc>. (Cited on page 189.)
- [23] Rob J Hyndman and George Athanasopoulos: *Evaluating forecast accuracy*, 2013. <https://www.otexts.org/fpp/2/5>. (Cited on page 121.)
- [24] J.DeGooijer and R.Hyndman: *25 years of time series forecasting*. International Journal of Forecasting 22 (3), 443-473. 2006. (Cited on page 118.)
- [25] Mehmet Ko and Atalay Barkana: *Application of linear regression classification to low-dimensional datasets, neurocomputing, volume 131, 5 may 2014, pages 331-335.*. (Cited on page 117.)
- [26] Lee and Brent Ware: *Open Source Development with LAMP: Using Linux, Apache, MySQL and PHP*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2002, 2002. (Cited on page 26.)
- [27] Juan J. Merelo, A. Prieto, F. Morán, R. Marabini, and J. M. Carazo.: *Automatic classification of biological particles from electron-microscopy images using conventional and genetic-algorithm optimized learning vector quantization*. Neural Processing Letters, 8 (1), 55-65. 1998. (Cited on page 114.)
- [28] E. Parras-Gutierrez, M. Garcia-Arenas, V. Rivas, and M. del Jesus: *Coevolution of lags and rbfns for time series forecasting: L-co-r algorithm*. Soft Computing 16 (6) (2012) 919-942. (Cited on page 116.)
- [29] E. Parras-Gutierrez, V.M. Rivas, M. Garcia-Arenas, and M.J. del Jesus: *Short, medium and long term forecasting of time series using the l-co-r algorithm*. Neurocomputing, Volume 128, 27 March 2014. (Cited on page 116.)
- [30] David A. Patterson, Garth Gibson, and Randy H. Katz: *A case for redundant arrays of inexpensive disks (raid)*, 1998. (Cited on page 25.)
- [31] Inés Gómez Plaza: *Estadísticas usuarios redes sociales en España año 2013*, 2013. <http://www.concepto05.com/2013/07/estadisticas-usuarios-redes-sociales-en-espana-2013/>. (Cited on page 188.)
- [32] Martin P.T., Y. Feng, and X. Wang: *Detector technology evaluation. technical report, utah transportation centre*. 2003. (Cited on page 4.)

- [33] J. Ross Quinlan: *C4.5: Programs for machine learning*. Morgan Kaufmann Series in Machine Learning. 1992. (Cited on page 114.)
- [34] J. Ross Quinlan: *Improved use of continuous attributes in c4.5*. *j. Artif. Intell. Res. (JAIR)* 4: 77-90. 1996. (Cited on page 114.)
- [35] R.Hyndman and A.Koehler: *Another look at measures of forecast accuracy*. *International Journal of Forecasting* 22 (4) 679-688. 2006. (Cited on page 118.)
- [36] V. Rivas, J. Merelo, P. Castillo, M. Arenas, and J. Castellano: *Evolving RBF neural networks for time-series forecasting with EVRBF*. *Information Sciences* 165 (3-4), 207-220. 2004. (Cited on page 114.)
- [37] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko: *High Performance MySQL: Optimization, Backups, and Replication*. O'Reilly. (Cited on page 28.)
- [38] Peter Snyder: *tmpfs: A virtual memory file system*. In *Proceedings of the Autumn 1990 EUUG Conference*, pages 241–248, 1990. (Cited on page 27.)
- [39] J. A. K. Suykens and J. Vandewalle: *Least squares support vector machine classifiers. neural process. lett.* 9, 3 (june 1999), 293-300. 1999. (Cited on page 117.)
- [40] H. Tong: *On a threshold model, pattern recognition and signal processing*. NATO ASI Series E: Applied Sc. 29 (1978) 575-586. (Cited on page 115.)
- [41] H. Tong: *Threshold models in non-linear time series analysis*. Springer-Verlag, 1983. (Cited on page 115.)
- [42] Ian Witten and Eibe Frank: *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 3 edition, 2011. (Cited on page 117.)
- [43] G. Zhang and M. Qi: *Neural network forecasting for seasonal and trend time series*. *European Journal of Operational Research* 160 (2) (2005) 501-514. (Cited on page 116.)

