
***Programa de Experto en Big Data***

*MEMORIA DE PROYECTO FINAL*

# *Análisis del uso del "bicing" de Barcelona*

*Alumno: Juan Rodríguez Hortalá*

*Tutor: Pere Ferrera Bertrán*

*Fecha: 27/07/14*

# Descripción

En este proyecto se han estudiado los datos sobre el uso de la red "Bicing" de bicicletas de uso compartido de la ciudad de Barcelona, ofrecida a través de la plataforma open data del ayuntamiento de Barcelona, para intentar detectar patrones de uso del sistema (estaciones más demandadas, patrones de movilidad, …) que pudieran utilizarse para optimizar el uso de la red Bicing.

*Contenidos entregados*

- Memoria en formato PDF.
- Código fuente en un solo archivo ZIP
- Documentación del código fuente: un solo archivo en formato ZIP con los archivos HTML correspondientes a la documentación del código.

# Objetivos

## Objetivos originales

Los objetivos originales recogidos en el Anteproyecto son los siguientes:

- Realizar visualizaciones en tiempo real del uso del sistema Bicing.
- Desarrollar algunos cubos OLAP para el análisis histórico de los datos del sistema Bicing.
- Opcionalmente, utilizar la base de datos SciDB para análisis OLAP, desarrollando un provider XMLA para SciDB.
- Utilizando los sistemas desarrollados, sacar conclusiones acerca del estado actual del uso del sistema Bicing. Estas conclusiones serán inevitablemente limitadas, por el desconocimiento del dominio, y por la falta de datos históricos. Por ello el objetivo principal del proyecto es desarrollar las herramientas técnicas que habilitaran análisis futuros por analistas con conocimiento del dominio pero sin grandes capacidades técnicas, más que el análisis de datos en sí mismo.

## Objetivos conseguidos

El sistema finalmente desarrollado durante el proyecto ofrece las siguientes funcionalidades relativas a los objetivos iniciales:

- Sistema escalable de ingesta en streaming, que permite la ingesta de datos procedentes de un varios servicios web, añadiéndose nuevos servicios mediante un sistema de ficheros de configuración. Este resultado cubre parcialmente el objetivo de las visualizaciones en tiempo real, y también se usa para la ingesta en el análisis OLAP.
- Desarrollo de un sistema de análisis OLAP, incluyendo el modelo multidimensional, la implementación de un proceso ETL en streaming, y la visualización mediante varios informes.
- Aunque no se han implementado, se ha realizado el análisis funcional para las visualizaciones en streaming, especificándose varias visualizaciones posibles que podrían ser de interés para la monitorización en tiempo real del uso del sistema

Bicing. También se han dado pinceladas de un posible análisis de series temporales.

El resto de objetivos no han podido cubrirse, en parte debido a lo ambicioso del proyecto, y también por limitaciones de tiempo por motivos laborales, y algunas limitaciones descubiertas en las tecnologías propuestas que han dificultado y retrasado el desarrollo.

# Metodología de desarrollo

El proyecto se desarrolló utilizando una versión ligera de la metodología Scrum, al tratarse de un proyecto de una sola persona. Se utilizaron los siguientes artefactos:

- Como repositorio de código se utilizó Github, en concreto el proyecto https://github.com/juanrh/bicing-bcn.
- Github también se utilizó para fijar algunos hitos para el proyecto mediante su sistema de Issues y Milestones.
- La wiki de Github también se utilizó para el análisis funcional, concretando los objetivos del proyecto en la página https://github.com/juanrh/bicing-bcn/wiki/Project-goals, y también como espacio de trabajo en el que tomar notas y apuntar referencias interesantes.
- El servicio web ScrumDo se utilizó para realizar una gestión de proyecto ligera utilizando Scrum, quedando registrados las tareas asignadas a cada sprint y medido automáticamente el ritmo de trabajo, lo que resultó muy útil para ayudar a detectar cuanto antes que la totalidad de los objetivos del proyecto no podrían alcanzarse a tiempo. La URL del proyecto ScrumDo es https://www.scrumdo.com/projects/project/bicing-bcn, aunque esta URL no es pública sí que se dió acceso al tutor del proyecto.

# Análisis del problema

Como cualquier proyecto relacionado con el análisis de datos, los datos de partida determinan qué información puede derivarse y por tanto qué preguntas pueden responderse, y qué problemas pueden resolverse utilizando esas respuestas.

## Datos de partida

Los datos de partida en este proyecto son los correspondientes al servicio web “Estaciones de Bicing” de la ciudad de Barcelona descrito en [1] y accesible desde la URL [2]. Este servicio proporciona un archivo XML [4] que se actualiza cada minuto aproximadamente, y que contiene la fecha de actualización como una marca de tiempo en Unix time [3] (en segundos), y un elemento XML por cada estación de la red Bicing con el siguiente formato:

```
<bicing_stations>
  <updatetime><![CDATA[1400940246]]></updatetime>
  <station>
    <id>1</id>
    <lat>41.3979520</lat>
    <long>2.18004200</long>
    <street><![CDATA[Gran Via Corts Catalanes]]></street>
```

```
    <height>21</height>
    <streetNumber>760</streetNumber>
    <nearbyStationList>24,369,387,426</nearbyStationList>
    <status>OPN</status>
    <slots>13</slots>
    <bikes>9</bikes>
  </station>
...
  </station>
</bicing_stations>
```

El significado de los campos del elemento XML `<station>` son los siguientes:

- `id`: identificador de la estación Bicing que corresponde a estos datos.
- `lat`, `long`: coordenadas de la estación.
- `street`, `streetNumber`: dirección de la estación, relativa a la ciudad de Barcelona.
- `height`: elevación de la estación respecto al nivel del mar.
- `status`: estado de la estación como en funcionamiento (`OPN` / open station) o estación cerrada (`CLS` / closed).
- `bikes`: número de bicicletas aparcadas en la estación en este momento, disponibles para ser prestadas.
- `slots`: número de huecos disponibles para aparcar bicicletas en esta estación. Este valor es relevante ya que es habitual que en un trayecto se aparque una bicicleta en una estación diferente a la de partida (por ejemplo al ir a trabajar), por lo que puede ser interesante para los usuarios saber cuál es la estación más cercana en la que queda sitio para aparcar. Inicialmente se pensó que este valor correspondía a la capacidad de la estación como número de bicicletas que pueden estar aparcadas a la vez, pero la descripción del servicio [1] confirma la definición anterior, y así lo hacen los datos, al haber múltiples archivos con un valor de `slots` inferior al de `bikes` para algunas estaciones.

Examinando los datos, se detectó también que la capacidad de las estaciones como el número de bicicletas que podrían aparcarse en una estación, es decir, `slots + bikes` si el estado es abierto y 0 en otro caso, cambia a menudo en los datos, con variaciones de +1 o -1 de un archivo XML al siguiente. Probablemente esto se explique porque algunos huecos de las estaciones dejan de estar funcionales y pasan a estarlo poco después, por mantenimiento por parte de operarios o por algún proceso automático de recuperación, o simplemente por algún tipo de ruido en los sensores.

## Análisis de la información disponible

A partir de los datos crudos anteriores podemos deducir distinta información derivada. También es importante ser conscientes de las limitaciones de los datos para saber que tipo de preguntas no podemos pretender responder.

### Información de la que no disponemos

- Localización de las bicicletas: no tenemos información de las bicicletas, solamente acerca de las estaciones.
- Identificador de bicicleta: las bicicletas son anónimas, y no hay forma de detectar la transferencia de una bicicleta de una estación a otra. Esto también se debe a que bicicletas no pertenecen a ninguna estación en concreto.

Nótese además que las bicicletas no se mueven de una estación a una de las estaciones cercanas, siendo habitual el saltarse varias estaciones en un trayecto medio.

**Información que podemos deducir**

- Evento de una bicicleta cualquiera entrando o dejando una estación: mediante la diferencia entre el conteo de bicicletas entre dos archivos XML consecutivos. Esta información es imprecisa porque no captura eventos como que se devuelva y preste la misma bicicleta entre dos actualizaciones, pero puede emplearse para calcular una estimación del tráfico en las estaciones.
- Podemos enriquecer la información espacial de las estaciones de diversas maneras:
    - Asociando el distrito, barrio y código postal correspondiente a las coordenadas y dirección de cada estación.
    - Obteniendo información adicional sobre la estación o distrito que intuitivamente pueda tener un impacto sobre el uso del sistema Bicing, como por ejemplo información relacionada con el turismo (número de monumentos en el distrito o a una cierta distancia de la estación, número de hoteles, …), información socio-económica del distrito (censo, sueldo medio, número de negocios, …)

## Modelo dimensional

Teniendo en cuenta esta información podemos definir las siguientes dimensiones y medidas para un modelo dimensional, que podemos utilizar de base tanto para el análisis OLAP como para definir las visualizaciones en streaming.

### Dimensiones

*Dimensión estación*: agrupa información propia de cada estación y que se mantiene estable a lo largo del tiempo.

- Información geográfica de la estación: longitud, latitud, altura.
- Metainformación de la estación: dirección e información derivada como distrito, barrio y código postal, que es más útil que la geográfica ya que permite agrupar varias estaciones de forma fácil sin necesidad de recurrir a capacidades de procesamiento de información geoespacial.

Nótese que se asume la integridad de el campo `id` de la fuente de datos, entendiéndose que el id de cada estación la identifica unívocamente.

*Dimensión estado de la estación*: información relativa a la estación que cambia frecuentemente, sólo incluye su estado como abierta o cerrada.

*Dimensión temporal*: como es habitual en OLAP, descompondremos las fechas estableciendo una estructura jerárquica de año, trimestre, mes, semana, día y otras divisiones temporales como sexto del día ([4-8), [8-12), [12-16), [16-20), [20,0), [0, 4)) o partes del día (ir a trabajar, hora de la comida, vuelta a casa, …), que servirá para realizar diversas agrupaciones de las medidas.

### Medidas

Consideramos las siguientes medidas, todas trabajando a nivel de estación:

- Medidas de *capacidad*: número de huecos disponibles para aparcar, número de bicicletas disponibles, número de estaciones. Para estos conteos conviene que se asuma un 0 para las estaciones que están cerradas, lo que se implementará como parte de la lógica de la ETL.
- Medidas de *tráfico*: número de bicicletas prestadas y devueltas, y tráfico como suma de ambas.

Para estas medidas consideraremos tanto sumas como medias, según convenga a la situación.

## Preguntas a responder

Estas dimensiones y medidas ofrecen una perspectiva del sistema Bicing, que podemos utilizar para plantear una serie de preguntas acerca del uso de Bicing, y también para responderlas por medio de diversas visualizaciones de los datos. Estas preguntas y respuestas se presentan a continuación, clasificadas por tipo de análisis: informes OLAP, visualizaciones en streaming, o análisis de series temporales.

### Informes OLAP

**¿Qué áreas de la ciudad se usan más frecuentemente como origen y destino?**

Se confeccionará un informe que muestre una barra apilada por estación con una parte por fragmento del día, para el número de bicicletas prestadas durante la última semana, ordenando las barras por el número de bicicletas prestadas. También se diseñará el informe equivalente para bicicletas devueltas.

Estos informes también pueden tener distintas variantes, por ejemplo agrupando las estaciones por distrito o por otros criterios como información socio-económica del distrito de la estación, o utilizando otras agrupaciones temporales como diario o mensual.

**¿Dónde debería situarse una nueva estación?**

La idea es que deberíamos situar una estación en un área con capacidad cercana al límite y con mucho tráfico. Por tanto, podemos definir una tabla con las siguientes filas: distrito, capacidad como número medio de bicicletas disponibles (0 cuando la estación está

cerrada), y tráfico como el número de bicicletas prestadas más el número de bicicletas devueltas. El informe devolverá las mejores diez áreas, ordenadas de manera ascendente por capacidad y descendente por tráfico; definiremos informes para la última semana y para el último mes.

**¿Cuáles son los patrones de uso de las estaciones a lo largo del día?**

Definimos una gráfico de línea para cada estación, con las horas en el eje horizontal y las siguientes líneas:

- Número medio de bicicletas disponibles en esa hora.
- Número medio de bicicletas prestadas en esa hora.
- Número medio de bicicletas devueltas en esa hora.

Este informe se puede obtener para el último día (de tal manera que la media actúa como la función identidad) o bien para la última semana o para el último mes. Este informe también podría agrupar estaciones por distrito o cualquier otro criterio, y también puede agrupar horas en partes del día.

**Otras cuestiones**

El usuario también puede definir nuevos informes usando la interfaz de usuario de Saiku.

### Visualizaciones en streaming

**¿Cuál es el estado actual de sistema de bicing?**

Se confeccionarán mapas de calor de tiempo real sobre CartoDB para:

- Estado de las estaciones (abierto/cerrado).
- Número de bicicletas disponibles para cada estación, tomando 0 para las estaciones cerradas.
- Media móvil simple durante la última hora para el número de bicicletas prestadas por estación.
- Media móvil simple durante la última hora para el número de bicicletas devueltas por estación.

También se desarrollarán gráficos en tiempo real sobre OpenTSDB correspondientes a los cuatro mapas anteriores:

- Gráfico de línea con una línea para el número de estaciones abiertas y otra para el número de estaciones cerradas.
- Gráfico de línea con una línea por distrito con el número de bicicletas disponibles en las estaciones de dicho distrito.
- Gráfico de línea con una línea por distrito con la media móvil simple durante la última hora para el número de bicicletas prestadas en las estaciones de dicho distrito.
- Gráfico de línea con una línea por distrito con la media móvil simple durante la última hora para el número de bicicletas devueltas en las estaciones de dicho distrito.

Otras posibles extensiones:

- Cómputo y visualización en una línea de referencia horizontal de la media móvil simple para toda la ciudad o para una periodo más largo (día, semana…) para cualquiera de las métricas anteriormente mencionadas.
- También sería interesante combinar información histórica con información en tiempo real. Para ello, un enfoque simple sería definir un trabajo Pig para computar la media durante el último mes y trimestre para aquellos tres valores y guardarlos en HBase. Este trabajo podría ejecutarse cada hora y se representaría como un par de líneas de referencia horizontales constantes en cada uno de estos gráficos.

### Análisis de series temporales

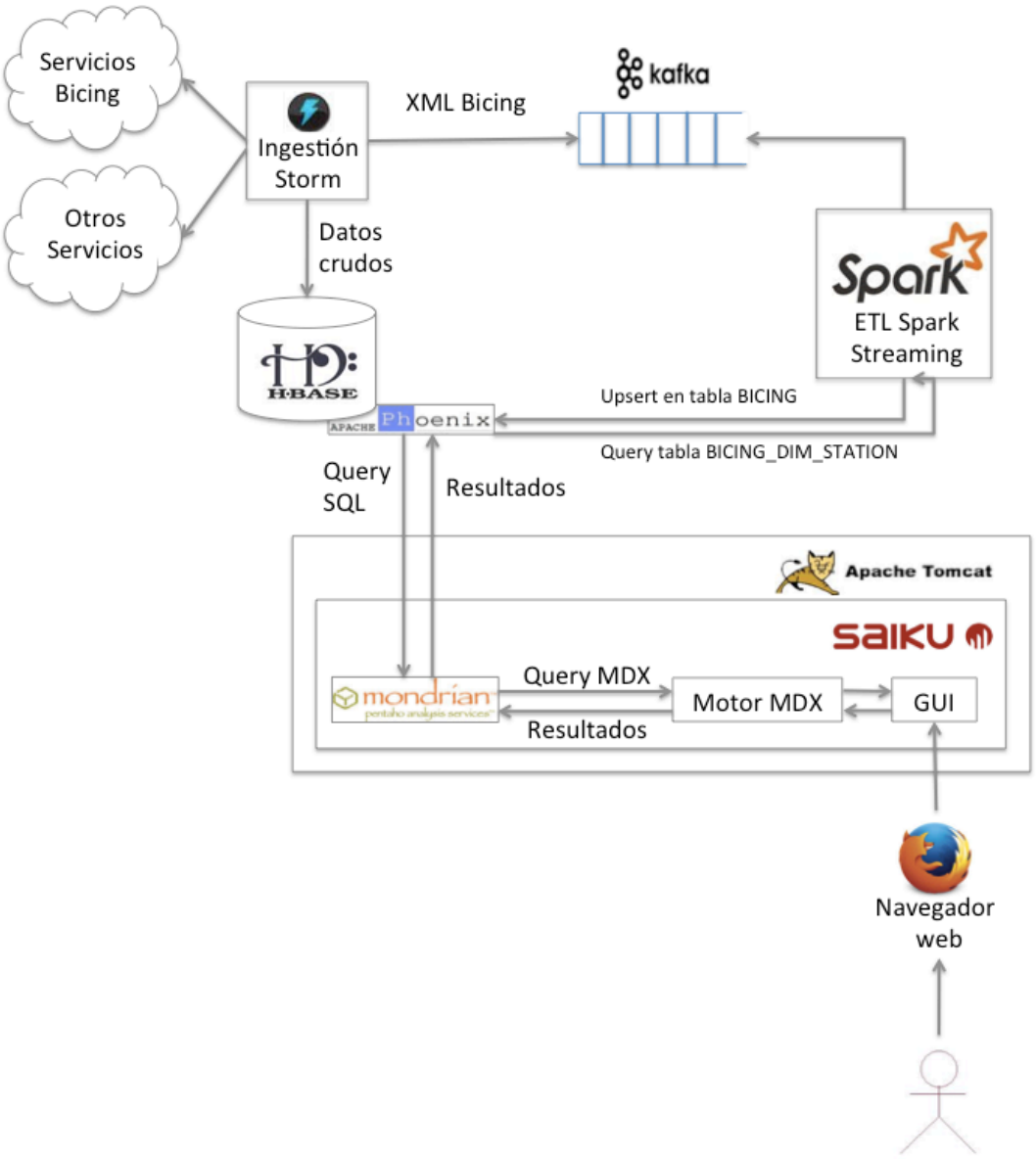
**¿Cuáles son las predicciones de la capacidad y el tráfico de las estaciones?**

Las predicciones de serie temporales se harán sobre las siguientes series temporales:

- Una variable por estación, un nuevo valor cada hora para el número medio de bicicletas disponibles durante esa hora.
- Una variable por estación, un nuevo valor cada hora para el número medio de bicicletas prestadas durante esa hora.
- Una variable por estación, un nuevo valor cada hora para el número medio de bicicletas devueltas durante esa hora.

## Diseño e implementación del sistema

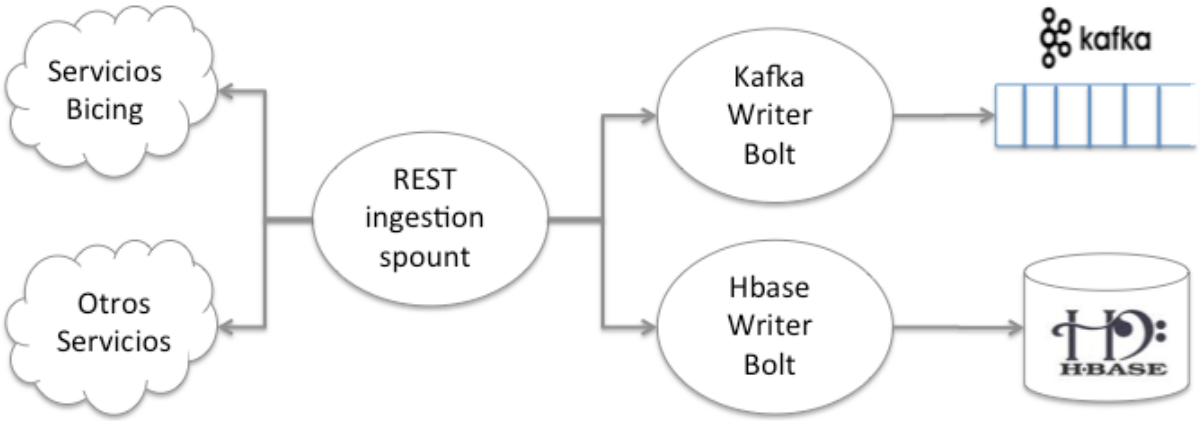
Como se indicó en la sección Objetivos sólo se ha llegado a implementar la ingesta de datos y el análisis OLAP. El siguiente diagrama da una visión de alto nivel de la arquitectura del sistema implementado.




- Para la ingesta de datos se ha empleado Storm, con el que se ha desarrollado un sistema genérico para el consumo de servicios web.
- El proceso ETL se ha realizado en Spark Streaming, que se comunica con Storm a través de Apache Kafka.
- El resultado de la ETL se almacena en HBase a través de Apache Phoenix, que añade capacidades SQL a HBase. Esto permite utilizar Saiku como front-end web (corriendo sobre Tomcat) para análisis OLAP, con el que interactúan los usuarios a través del navegador.

## Ingesta de datos

La ingesta de datos se ha implementado como la siguiente topología Storm.




Este sistema es genérico al poder utilizarse para consumir diferentes servicios. Cada servicio se registra en el sistema añadiendo un archivo de configuración para el servicio en una ruta determinada. Al arranque de la topología, Storm comprueba esa ruta y configura el número de workers, y de tareas de cada componente de la topología en función del número de fuentes de datos.

Cada instancia del spout se encarga de unas fuentes de datos fijas, a las que consulta con la frecuencia configurada en el fichero correspondiente. Para ello se mantiene en memoria un estado para cada fuente de datos, que incluye un contador de tiempo, para saber cuando ha llegado el momento de pedir más datos a la fuente. Se ha utilizado el mecanismo de procesamiento garantizado de Storm, utilizándose un UUID aleatorio para identificar a cada tupla, y Redis a modo de HashMap local a cada instancia del spout pero con snapshots periódicos en disco proporcionados por la persistencia RDB por defecto de Redis.

Cada dato obtenido de un servicio se almacena en crudo en HBase en una tabla para la fuente de datos, para tenerlos disponibles para posibles análisis adicionales. Esta es una manera sencilla de poder almacenar grandes volúmenes de archivos pequeños (una debilidad conocida de HDFS [8]), sin tener que preocuparse de consolidarlos más adelante en archivos grandes por medio de herramientas [7], al ocuparse de esto implícitamente el motor de HBase, que no impone un límite al tamaño de las celdas que puede alojar. En cualquier caso, se parte de la base de que cada actualización de los servicios manda archivos bastante pequeños, esta es una limitación de este sistema pero se entiende que es razonable para un sistema en streaming.

Finalmente cada dato se envía a Kafka a un topic dedicado a cada fuente de datos. Esto desacopla la ingesta de datos de su consumo en streaming, y permite procesar cada fuente de datos de forma independiente.

## Análisis OLAP

La implementación de análisis OLAP se basa en la utilización de Apache Phoenix [5] para añadir capacidades SQL a HBase, para poder entonces utilizar HBase como una base de datos analítica distribuida. Phoenix proporciona estas capacidades de dos maneras:

- Proporcionando un jar con definiciones de coprocessors HBase adicionales, que se instalan en todos los Region Server del cluster HBase. Estos coprocessors se utilizan para poder realizar operaciones de agregación como sumas, conteos y medias, de forma distribuida en HBase, y también para implementar triggers con los que se pueden mantener índices secundarios simulados sobre HBase.

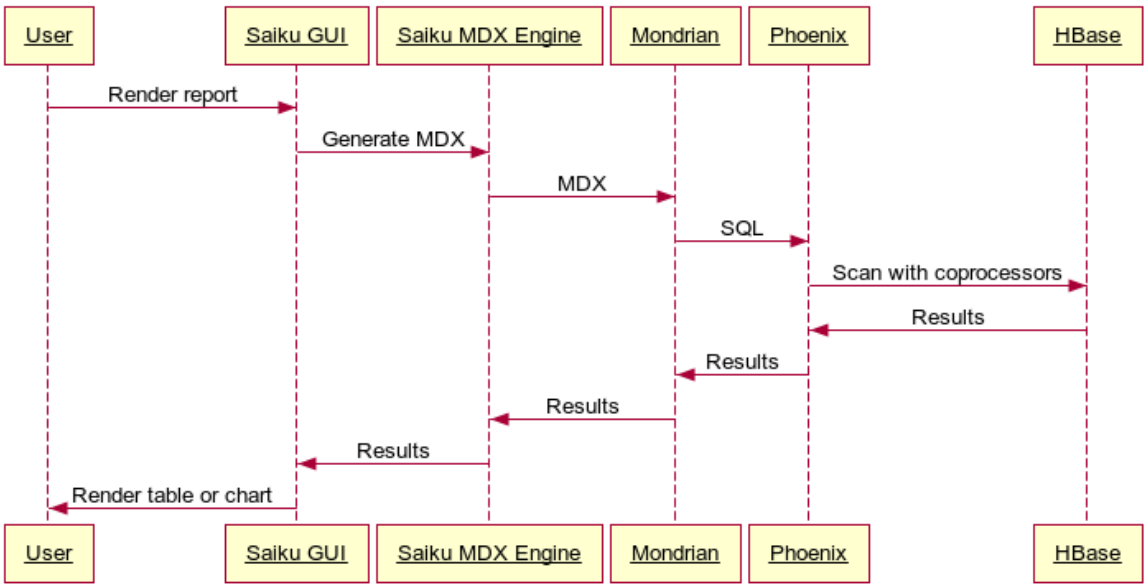
- También proporciona un cliente JDBC que se puede utilizar para hacer consultas SQL desde Java hacia HBase.

A partir de estas capacidades SQL añadidas a HBase, que permiten calcular agregaciones de datos distribuidas sobre el cluster HBase, se ha diseñado un sistema de análisis ROLAP (Relational OLAP) [9] basado en el front-end OLAP Saiku [10] y en la integración de este con el servidor OLAP Mondrian [11]. El uso de Mondrian con Saiku es bastante común, jugando estos dos componentes los papeles siguientes:

- Mondrian permite definir un cubo OLAP a nivel lógico y su mapeo físico a tablas de una base de datos relacional. A partir de ello Mondrian es capaz de transformar consultas MDX [12], el lenguaje de consultas nativo de los cubos OLAP, en consultas SQL que pueden lanzarse sobre la base de datos relacional que soporta físicamente el cubo OLAP.
- Sobre Saiku se configura una fuente de datos Mondrian, que Saiku representa gráficamente en su front-end web. El usuario puede generar informes sobre el front-end, que Saiku convierte en queries MDX que son enviadas a Mondrian para que éste las resuelva.

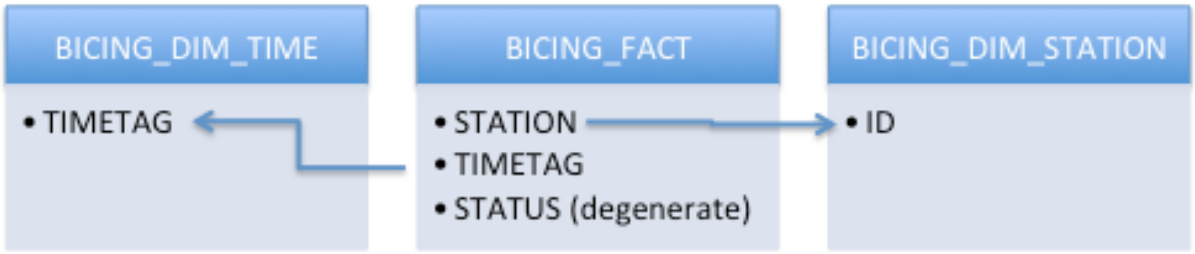
La cuestión es que Mondrian en realidad no necesita que la base de datos que da soporte físico a los cubos OLAP sea relacional, sino que simplemente requiere un driver JDBC con soporte para SQL para conectarse a la base de datos. Por tanto una de las hipótesis a validar en el presente proyecto ha sido comprobar si Phoenix proporcionaba un soporte SQL lo bastante completo como para ser utilizado por Mondrian. Como veremos más adelante la respuesta es que sí, pero con matices y ciertas limitaciones.

La integración de todos los componentes se resume en el siguiente diagrama, donde Phoenix sirve de puente entre Mondrian y HBase, extendiendo el montaje habitual de colaboración entre Saiku y Mondrian.



### Esquema de estrella en Phoenix

El modelo dimensional a implementar en un cubo OLAP para el análisis del sistema Bicing es el definido en la sección **Modelo dimensional**. Este modelo lógico debe mapearse a un esquema relacional concreto sobre Phoenix para su uso desde Mondrian. Inicialmente la idea era utilizar el siguiente esquema de estrella [6].




En este esquema, para la dimensión estación se considera suficiente la estrategia slow changing dimensions de tipo 1 [11], al asumir que la información sobre las estaciones es muy estable a lo largo el tiempo. Por otra parte la dimensión de estado de la estación se trata como una dimensión degenerada [6] asumida en la tabla de hechos, al tratarse de una dimensión compuesta de un único campo.

La ventaja de utilizar un esquema en estrella es que permite que todos los registros de la tabla BICING_FACT para la misma estación compartan el mismo registro para la tabla BICING_DIM_STATION, y lo mismo ocurre para el campo del código de tiempo y la tabla BICING_DIM_TIME. Esto permite un gran ahorro en disco y red, y su impacto en la velocidad de las consultas es muy bajo debido a las optimizaciones específicas para consultas sobre esquemas en estrella. Estas se basan en:

- Los esquemas en estrella restringen a que todas las tablas de dimensiones (BICING_DIM_STATION, BICING_DIM_TIME) estén como mucho a un join de distancia de la tabla de hechos (BICING_FACT), por lo que se evitan joins encadenados.
- Las tablas de hechos suelen ser estrechas y largas (muchos registros pero pocas columnas), mientras que las tablas de dimensiones suelen ser anchas y cortas (pocos registros y mucha columnas). Esto se ejemplifica bien en la tabla BICING_DIM_STATION, en la que asumimos que el número de estaciones es pequeño y estable a lo largo del tiempo. Por tanto al hacer estos joins se pueden guardar en memoria las tablas de hechos (al estilo de un memory-map join), como se hace por ejemplo en Apache Phoenix [13].

Sin embargo este esquema en estrella no se ha podido implementar, debido a que Phoenix (al menos en su versión 3.0.0 utilizada en el proyecto, que es la versión compatible con HBase 0.94.6 distribuida con CDH4) no soporta joins implícitos, lanzando excepciones como la siguiente en respuesta a la consulta SQL generada por Mondrian para el esquema en estrella anterior:

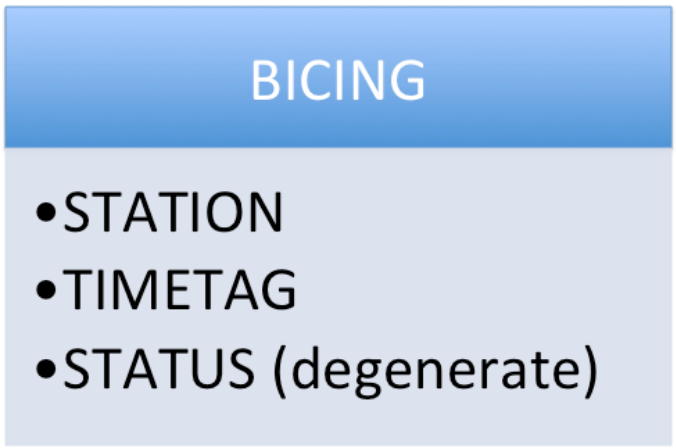
```
SQLFeatureNotSupportedException: Implicit joins not supported.

Caused by: mondrian.olap.MondrianException: Mondrian Error:Internal error:
Error while loading segment; sql=[select "BICING_DIM_STATION"."DISTRICT" as
"c0", sum("BICING_FACT"."AVAILABLE") as "m0" from "BICING_DIM_STATION" as
```

```
"BICING_DIM_STATION", "BICING_FACT" as "BICING_FACT" where
"BICING_FACT"."STATION" = "BICING_DIM_STATION"."ID" group by
"BICING_DIM_STATION"."DISTRICT"]
```

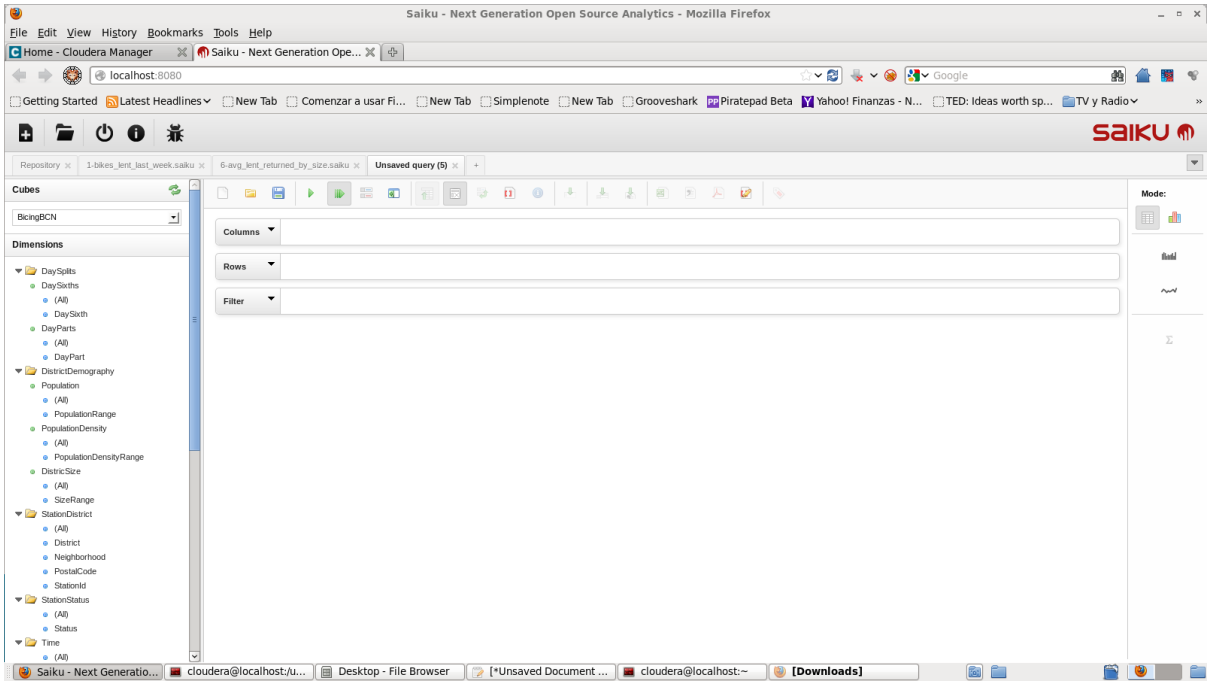
Como puede verse en la consulta incluida en el error, en este tipo de joins implícitos no se utiliza explícitamente el comando `JOIN`, sino que es el motor de consultas SQL el que debe deducir que un join es necesario al aparecer en el `WHERE` una condición de igualdad entre dos campos de tablas que aparecen en el `FROM`. Por tanto, a pesar de que Phoenix soporta joins de forma eficiente [13], esa característica no pudo explotarse en este proyecto debido a la falta de flexibilidad que implica utilizar Phoenix a través de Mondrian.

La decisión que se tomó para superar este problema ha sido denormalizar completamente el esquema, fusionando las dos tablas de dimensiones con la tabla de hechos en una sola tabla (tablón) BICING, que contiene para cada registro toda la información acerca de una actualización de la estación: tanto la información nueva proporcionada por el servicio Bicing como la información propia de la estación, y la información de tiempo derivada de la fecha de actualización.

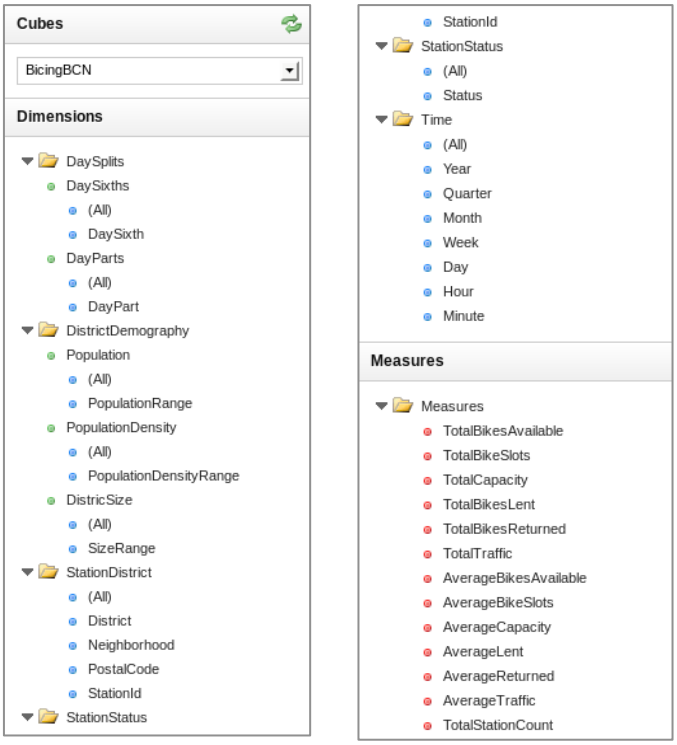



La principal desventaja de este enfoque es que se replica la información de cada estación en todos los registros relacionados con la estación. Lo mismo ocurre con la información de tiempo, pero esto tiene menos impacto porque cada registro de tiempo sólo se replica tantas veces como estaciones hay en la red Bicing. De hecho incorporar BICING_DIM_TIME dentro de BICING_FACT como una serie de dimensiones degeneradas hubiera sido otra opción a valorar aún el caso de que las joins de Phoenix fuesen compatibles con Mondrian.

Una vez configurados Saiku, Mondrian y Phoenix, el resultado visible para el usuario es la interfaz web de Saiku, en la que podemos ver la definición del cubo OLAP en la columna de la izquierda, y componer informes arrastrando dimensiones y medidas al cuadro central [10].




En las siguientes figuras podemos ver el detalle del cubo OLAP implementado, donde pueden apreciarse dimensiones adicionales definidas como intervalos temporales o sobre las características de los distritos de las estaciones.

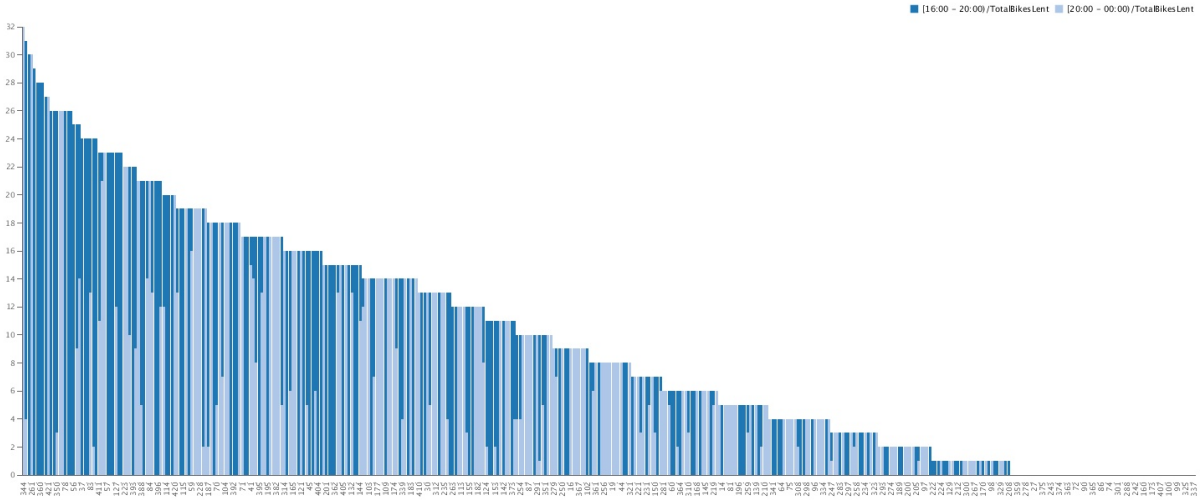



Sobre este montaje se desarrollaron los informes especificados en la sección **Informes OLAP**. Debido a la limitada capacidad de cómputo de la máquina virtual de desarrollo, los

informes siguientes se desarrollaron sobre pocos datos, lo que hace que el rango de valores que aparece en ellos no incluye a todos los valores posibles

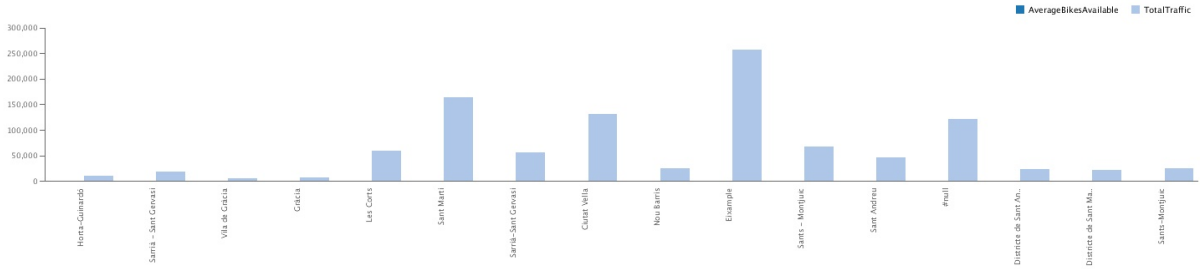
**¿Qué áreas de la ciudad se usan más frecuentemente como origen y destino?**

En este informe aparecen una barra por estación con el número de bicicletas prestadas en el último mes, partiendo la barra por parte del día y ordenando de mayor a menor número de bicicletas prestadas.



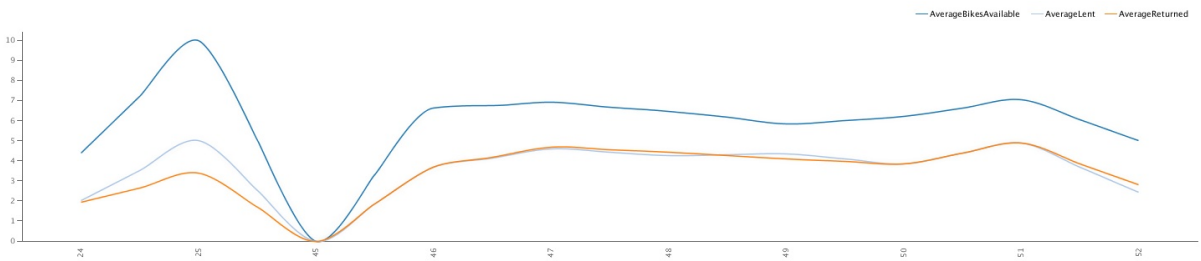

**¿Dónde debería situarse una nueva estación?**

En este informe se muestra la media de bicicletas disponibles y la media de tráfico en el último mes por distrito.

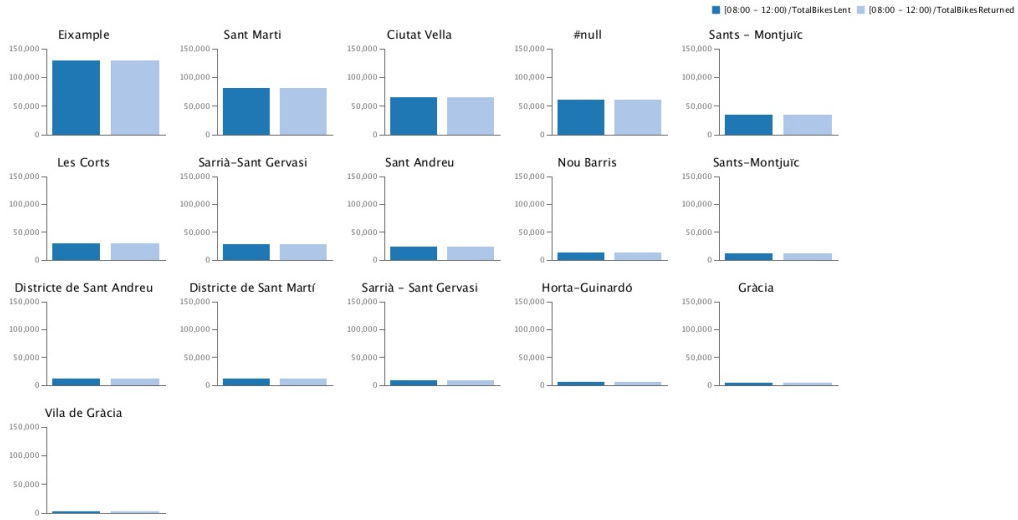



**¿Cuáles son los patrones de uso de las estaciones a lo largo del día?**

Aquí mostramos la media de bicicletas disponibles, prestadas y devueltas para una estación determinada, durante la última hora para una estación concreta.



También podemos hacer otro tipo de informes como el siguiente, que muestra el total de bicicletas prestadas y devueltas por distrito y por parte del día:




Se encontraron algunas dificultades durante el desarrollo de estos informes, debidas fundamentalmente a ciertas limitaciones inesperadas de Saiku como front-end BI. En concreto:

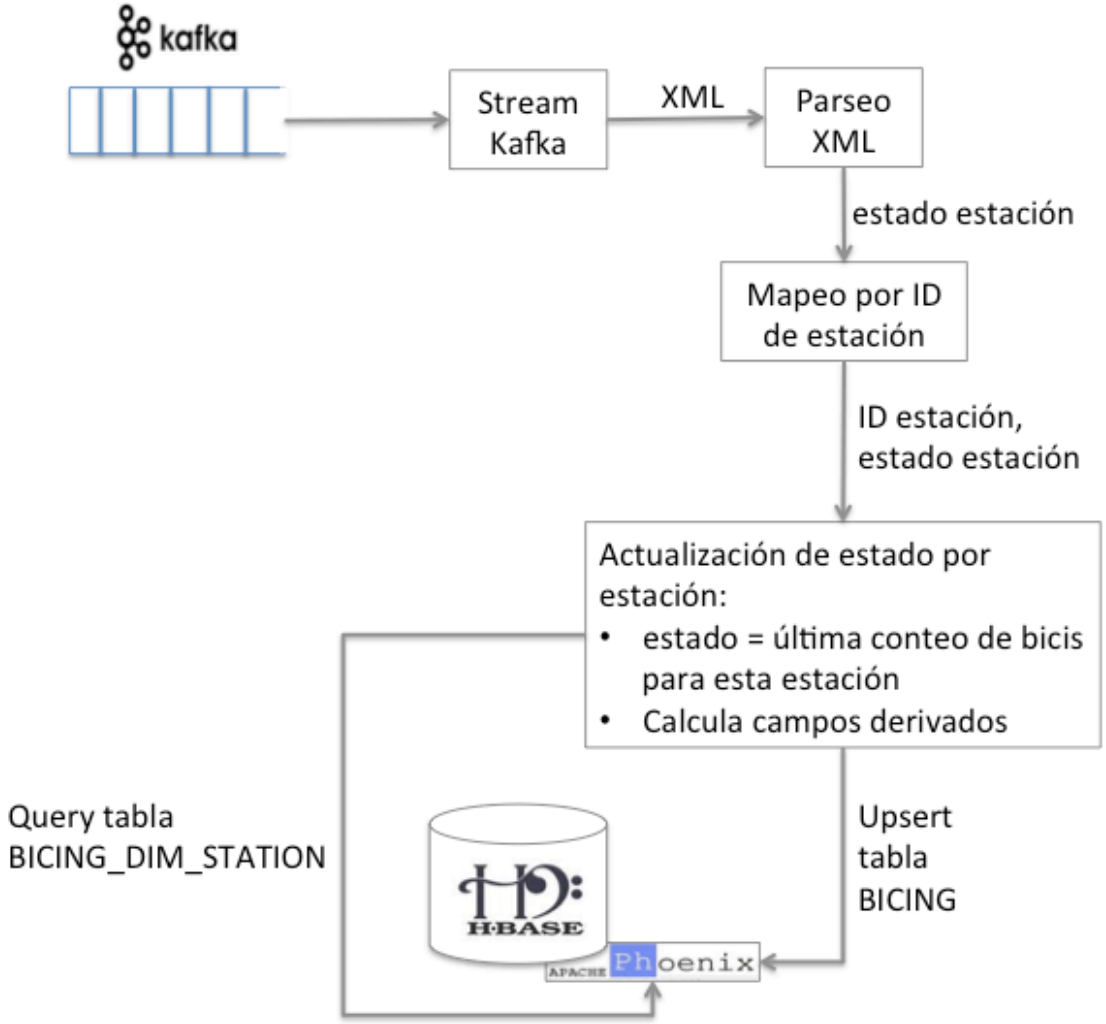
- No es posible filtrar los resultados para el último mes o la última semana (o cualquier otro periodo temporal fijo) desde el interfaz gráfico de Saiku.
- Tampoco es posible ordenar por más de un campo desde la GUI de Saiku.

Estas limitaciones se pueden superar introduciendo directamente la consulta MDX en Saiku, quedando la funcionalidad de Saiku limitada a la comunicación con Mondrian y las visualizaciones, que no es poco. Además Saiku puede usarse para definir un primer esqueleto de query MDX sobre el que trabajar. No obstante este imprevisto retrasó el desarrollo ya que hizo falta modificar a mano las consultas MDX de varios informes.

Por otra parte una última funcionalidad de Saiku es que puede utilizarse como repositorio de informes, de modo que los informes desarrollados puedes guardarse en Saiku y consultarse después, persistiendo entre sesiones de Saiku e incluso despliegues de Tomcat (los informes se guardan en formato XML y se encuentran en la carpeta bicing-bcn/OLAP/saiku/saiku-repository del código del proyecto y en la ruta correspondiente del repositorio Github). Gracias a esto, sería factible un escenario en que un usuario de negocio pudiera consultar un informe Saiku basado en MDX complejo, sin necesidad de que el usuario tuviera conocimientos de MDX.

## ETL

Para alimentar el tablón BICING descrito en la sección anterior se implementó un proceso ETL en Spark Streaming, que parte de los datos publicados en Kafka por la ingesta en Storm, y realiza el proceso descrito en el siguiente diagrama.




El proceso es bastante sencillo. Primero se parsea el XML procedente del servicio web de Bicing y se genera un valor como un POJO para cada estación. Estos valores se agrupan por estación (el servicio Bicing emite los datos agrupados por fecha de actualización) para poder aplicar una operación de actualización basada en llave y estado. En este caso la llave es el id de estación y el estado es el número de bicicletas disponibles, necesario para calcular las transacciones de prestar y devolver bicicletas. También se calculan campos derivados, que pueden caer en dos categorías:

- Datos de la dimensión tiempo: se calculan todos a partir del timetag de la actualización.
- Datos de estación: para esto se accede a la tabla de la dimensión estación definida en el esquema de estrella.

Con estos datos derivados se genera un nuevo registro a insertar en el tablón BICING, ejecutando una denormalización al vuelo. Todo este proceso es idempotente, en particular por el uso de UPSERT en la actualización a Phoenix, por lo que es compatible con el modelo de tolerancia a fallos de Spark Streaming, ya que en caso de tener que relazarse el procesamiento de algún registro el resultado obtenido sería el mismo.

### Enriquecimiento de la dimensión estación

Para construir la tabla BICING_DIM_STATION se ha desarrollado un programa Python (ver bicing-bcn/OLAP/phoenix/update_stations_dimension.py en el código) que partiendo de un archivo XML de Bicing de ejemplo construye un script SQL con un UPSERT por estación, generado a partir del siguiente proceso para cada estación:

- Se intenta detectar el distrito, barrio y código postal de la estación mediante geocoding inverso atacando a Google Maps a partir de las coordenadas de la estación, y luego a Wikipedia en caso de que Google Maps no obtenga toda la información.
- Se consulta el distrito en Wikipedia y se parsea el resultado para buscar el tamaño, población y densidad de población del distrito.

Este proceso es bastante imperfecto, y por ejemplo puede llegar a generar nombres ligeramente diferentes para el mismo distrito para dos estaciones del mismo distrito, pero es capaz de generar un primer borrador de tabla de información de estaciones que puede ser repasado y corregido por un humano con conocimiento de la ciudad de Barcelona.

## Escalabilidad de la solución

Se puede afirmar que el sistema resultante es escalable porque todos los componentes en los que se basa son escalables de forma horizontal.

- La ingesta se basa en Storm, HBase y Kafka, todos sistemas distribuidos y escalables, y a nivel de programa se trabaja a nivel de mensaje recibido de la fuente de datos, que se entiende es razonablemente pequeño al tratarse de una fuente en tiempo real.
- La ingesta se conecta con la ETL a través de Kafka, y utiliza Spark Streaming como motor de procesamiento distribuido. Aunque Spark Streaming está menos maduro que Storm, se basa en Spark que es un motor de procesamiento escalable. A nivel de programa también se trabaja como mucho con mensajes de la fuente de datos, o con pequeñas agrupaciones de registros para una misma estación. Finalmente el resultado se almacena en HBase.
- Las queries OLAP se resuelven en HBase de forma distribuidas a través del mecanismo de coprocessors. Apache Phoenix es el componente menos maduro y cuya escalabilidad es más discutible, pero al menos el diseño se basa en un mecanismo ya presente en HBase, y la comunidad de Apache Phoenix afirma que su sistema es escalable. El hecho de haberse convertido en proyecto Apache top level en tan poco tiempo da cierto crédito a Phoenix, pero hubiera sido deseable haber hecho un estudio de rendimiento en su aplicación a ROLAP dentro de este proyecto. No obstante no se ha dispuesto de recursos hardware para ello, por lo que no ha podido realizarse.

Finalmente, hay que admitir que el sistema sólo se ha probado en una máquina virtual de desarrollo, por lo que queda pendiente bastante trabajo de ajuste fino de los parámetros de los diversos componentes, para conseguir un sistema con escalabilidad robusta y alto rendimiento.

Por otra parte el uso tanto de Storm como de Spark Streaming, aunque motivado por la curiosidad académica, podría ser viable en un entorno basado en YARN [15], debido a que Spark ya está disponible sobre YARN y en el caso de Storm está en vías de estarlo.

# Instalación y uso

El sistema se ha desarrollado sobre Cloudera CDH4 y depende de los siguientes componentes:

- Apache Storm 0.9.1-incubating
- Apache Spark 1.0.0 for CDH4
- Apache Kafka 0.8.1.1 on Scala 2.10
- Saiku Server 2.5 con Mondrian 3.5.7 (incluye servidor Tomcat de pruebas)
- Redis 2.4.10
- Apache Phoenix 3.0.0-incubating sobre HBase 0.94.6

La parte desarrollada en Python utiliza Python 2.7 y las librerías Requests, Boto, Twisted, lxml, pygeocoder, wikipedia y BeautifulSoup, todas disponibles en pip.

Para instalar el sistema basta con instalar esas dependencias y configurar Saiku utilizando los scripts disponibles en el código en la ruta bicing-bcn/OLAP/saiku. Las tablas Phoenix se inicializan con los scripts de bicing-bcn/OLAP/phoenix

Para poner en marcha el sistema basta con arrancar Saiku con el script start-saiku.sh disponible en su carpeta de instalación, lanzar la topología Storm de ingesta y el proceso ETL en Spark Streaming. La GUI de Saiku queda disponible en el puerto 8080 de la dirección en que se haya levantado Saiku, que es una aplicación Tomcat que puede desplegarse en cualquier servidor de aplicaciones compatible. El login por defecto de Saiku es admin - admin.

# Conclusiones

En este trabajo se ha desarrollado una herramienta para el análisis del uso del sistema Bicing de Barcelona basado en un sistema de informes OLAP con ingesta y ETL continua. También se ha hecho el análisis funcional de un posible sistema de visualización en tiempo real que se no ha llegado a implementar por falta de tiempo, y se ha dado alguna idea para un posible análisis de series temporales.

## Tecnologías empleadas

Aparte de las tecnologías listadas como dependencias en la sección Instalación y uso se han utilizado las siguientes librerías Java:

- Apache HttpComponents para el consumo de servicios web.

- Google Guava para la manipulación de colecciones, cachés en memoria, y para algunos idioms de estilo funcional combinado con Fugue, que resultan más naturales en el contexto de Spark.
- Google Autovalue para la generación automática de POJOs.
- Apache Metamodel para el parseo de XML, como experimento para el tratamiento de documentos XML como si fueran tablas relacionales.

## Limitaciones técnicas encontradas

Las mayores dificultades que se han encontrado en el desarrollo, aparte de las naturales al trabajar con tecnologías desconocidas para el desarrollador, son la rigidez de Saiku y Mondrian y las limitaciones del soporte SQL de Phoenix. Basar el sistema en estas tecnologías por una parte es una gran ventaja porque permite reutilizar mucho trabajo, pero por otra parte es una debilidad porque te limita a lo que estas tecnologías son capaces de hacer, y hace opacos gran parte de los procesos que realiza el sistema: los realizados por estos componentes.

Utilizar MDX crudo para superar la rigidez de Saiku es una solución general, pero implica aprender una tecnología compleja y con un futuro incierto, y además hace que Saiku pierda una parte importante de su valor. Por ello quizás se podría plantear explorar las capacidades de Pentaho BA en una posible extensión del proyecto. Pentaho BA ya tiene a Saiku como uno de sus plugins bandera, pero es probable que haya otros plugins en Pentaho BA que sean capaces de trabajar con Mondrian, al ser Pentaho el principal patrocinador del desarrollo de Mondrian.

Por otra parte utilizar un modelo de tablón denormalizado como schema Phoenix no es una solución demasiado elegante en mi opinión. Valdría la pena probar la versión 4 de Phoenix, disponible de serie en HDP 2.1. y que depende de HBase 0.98 para ver si soporta un dialecto SQL más rico. También se podría evaluar si la versión 4 de Mondrian es más compatible con Phoenix. Si a pesar de todo no pudiera volverse al esquema de estrella, podría estudiarse si el uso de las column families de HBase para implementar algo similar a las proyecciones de Vertica y otras bases de datos columnares [14] pudiera aumentar el rendimiento de las consultas. Esto es factible porque las column families de HBase son visibles desde los esquemas Phoenix, como un prefijo de los nombres de columnas que siguen siempre el formato "<column family>.<qual>".

También se utilizó Avro para la ingesta pero finalmente se descartó su utilización debido a que el enfoque empleado era incorrecto, y se sustituyó el bolt correspondiente por el bolt que escribe en HBase. La idea inicial era utilizar un archivo Avro en HDSF por fuente de datos y mes, para evitar el problema de crear muchos archivos pequeños en HDFS. Se llegó a desarrollar un bolt (ver bicing-bcn/storm/ingestion/src/main/java/org/collprod/bicingbcn/ingestion/attic/AvroWriterBolt.java en el código fuente) que era capaz de abrir el archivo correspondiente en modo append cuando este ya existía, pero al correr el programa en distribuido se detectó que el archivo se corrompía cada vez que se detenía la topología. El motivo es que como indica la documentación de Storm en http://nathanmarz.github.io/storm/doc-0.8.1/backtype/storm/task/IBolt.html#cleanup la invocación del método `cleanup()` no está garantizada en despliegues distribuidos, y por tanto no lo está en despliegues en producción:

"*Called when an IBolt is going to be shutdown. There is no guarentee that cleanup will be called, because the supervisor kill -9's worker processes on the cluster.*

*The one context where cleanup is guaranteed to be called is when a topology is killed when running Storm in local mode.*"

Al no llamarse a `cleanup()` no se cerraba el archivo y este quedaba corrupto. Por tanto el enfoque de un archivo Avro por fuente de datos y mes era incorrecto. Llego a plantearse la idea de usar una tupla especial al estilo de las tick tuples para avisar a un bolt de que debe invocar `cleanup()`, de forma que un spout dedicado estuviera esperando a aceptar un mensaje de pre-shutdown para hacer un broadcast de esta tupla especial a toda la topología, y luego escribir un script bash de parada en dos tiempos, que primero enviara el mensaje al spout de pre-shutdown y luego matara la topología usando el comando `storm`. Pero esa solución aparte de algo retorcida seguía basándose en la idea de que el bolt Avro se pasara un mes seguido ejecutándose sin interrupciones en el escenario nominal. Eso implicaría una tolerancia a fallos malísima para el sistema de ingesta de datos crudos. Por otra parte la solución basada en HBase es simple y asegura los datos con granularidad de la unidad de datos obtenida de la fuente, a base de delegar en HBase, por lo que se abandonó la vía basada en Avro.

En cuanto a Apache Metamodel, ha sido un experimento interesante y es una librería fácil de usar, pero el parseo XML es bastante lento. Probablemente esa lentitud se deba a estar usando el backend DOM de Metamodel, por lo que convendría experimentar con la versión SAX, que probablemente sea más rápida pero que es más difícil de emplear porque tiene un mapeo menos natural al modelo relacional que en mi opinión está subyacente a Metamodel.

Finalmente en cuanto a los datos, en el análisis OLAP se han descartado las coordenadas geográficas de las estaciones para el cubo OLAP, porque no pueden explotarse desde un sistema sin capacidades GIS como Phoenix o simplemente Mondrian y Saiku. No obstante esta información hubiera sido muy útil de completarse la integración con CartoDB. En este sentido se avanzó algo de trabajo (ver bicing-bcn/spark/stream-visuals/src/main/java/org/collprod/bicingbcn/heatmap/HeatmapStream.java en el código) pero se encontraron dificultades porque el servicio denegaba las peticiones cuando se generaban con una frecuencia demasiado alta. Queda la impresión de que a menos de que se disponga de un ancho de banda de peticiones garantizado bastante alto, el uso de servicios externos es muy difícil en un procesamiento en streaming, y es mejor limitarlo a procesos batch. Otro ejemplo de esto fue la construcción de la tabla BICING_DIM_STATION con el script Python, que después de un rato empezaba a fallar al llegarse al límite de peticiones gratuitas en los servicios de Google y Wikipedia, al tratarse de en torno a 400 estaciones. En este caso concreto podría modificarse el script para que cachee la información de cada distrito, reduciendo el número de peticiones.

# Bibliografía


1. Descripción del servicio web "Estaciones de Bicing" de la ciudad de Barcelona, http://opendata.bcn.cat/opendata/cataleg/TRANSPORT/bicing/
2. Servicio web "Estaciones de Bicing" de la ciudad de Barcelona, http://wservice.viabicing.cat/getstations.php?v=1
3. Unix time (Wikipedia), http://en.wikipedia.org/wiki/Unix_time
4. Extensible Markup Language (XML), http://en.wikipedia.org/wiki/XML
5. Apache Phoenix, "We put the SQL back in NoSQL", http://phoenix.apache.org/
6. Star Schema The Complete Reference, Christopher Adamson, McGraw Hill Professional, Jul 7, 2010.
7. Hadoop Archive: File Compaction for HDFS, Yahoo Hadoop Blog, https://developer.yahoo.com/blogs/hadoop/hadoop-archive-file-compaction-hdfs-461.html
8. The Small Files Problem, Cloudera Blog, http://blog.cloudera.com/blog/2009/02/the-small-files-problem/
9. ROLAP (Wikipedia), http://en.wikipedia.org/wiki/ROLAP
10. Saiku http://meteorite.bi/saiku
11. Mondrian in Action, Mondrian in Action, Open source business analytics. William D. Back, Nicholas Goodman, and Julian Hyde, September 2013
12. MDX (Wikipedia), http://en.wikipedia.org/wiki/Multidimensional_Expressions
13. Apache Phoenix star-join optimization, http://phoenix.apache.org/joins.html
14. The Power of Projections, http://www.vertica.com/2011/09/01/the-power-of-projections-part-1/
15. Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2, Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, Jeff Markham; Addison-Wesley Data & Analytics Series; 2014