Universitat Rovira i Virgili

Integración de un problema de IoT con soluciones de BigData

María José López Osa

TRABAJO FINAL DE MÁSTER

Iván Rodero Castro

MÁSTER EN INGENIERÍA COMPUTACIONAL Y MATEMÁTICAS

Bilbao
2017


Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © 2017 María José López Osa.

Permission is granted to copy, distribute and/or modify this document under the terms of the

GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

© (el autor/a)

Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

FICHA DEL TRABAJO FINAL de MÁSTER

Título del trabajo:	Integración de un problema de IoT con soluciones de BigData
Nombre del autor:	María José López Osa
Nombre del director:	Ivan Rodero Castro
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final de Máster:	Computación de altas prestaciones
Titulación:	*Máster en Ingeniería Computacional y Matemáticas*

Resumen del Trabajo (máximo 250 palabras):

Cada vez es más común escuchar hablar de Internet of Things, Big Data, o tratamiento de datos en tiempo real, aunque como aspectos separados de lo que a mi entender es un mismo contexto: interconexión de aspectos cotidianos para dar solución a problemas diarios en las vidas de las personas y las ciudades.

Con este concepto en mente, el principal objetivo de este proyecto ha sido integrar en un mismo sistema todas las fases por las que pasa la información desde que se genera hasta que se vuelve a almacenar una vez utilizada. Los distintos subsistemas integrados han sido un sistema distribuido de transferencia continua de datos (kafka), una plataforma de computación en tiempo real distribuida (Storm) y finalmente un sistema de almacenamiento distribuido también (hdfs).

Una vez establecida la plataforma, implanté un caso de uso en ella, consistente en generar datos simulando medidas de viento recogidas en dos ciudades, transmitirlas al sistema, eliminar las medidas mayores de 50 m/s dejando constancia de la traza y del número de medidas eliminadas, calculando la latencia en el momento de almacenaje en local, y además almacenando estos datos en un sistema hdfs.

Se establece por tanto un flujo de los datos desde su generación hasta su almacenamiento, pasando por una serie de procesos de los que se puede obtener a su vez información que se incluye en el sistema.

Abstract (in English, 250 words or less):

It is increasingly common to hear about Internet of Things, Big Data, or data processing in real time, but as separate aspects of what, in my opinion, is the same context: interconnection of regular aspects to solve daily problems in the lives of people and cities.

In order to make this concepts run together , the main objective of this project has been to integrate in the same platform all the sub-systems through which the information passes since the moment it is generated until it is stored once it is used and processed. These following systems have been integrated to build the final platform, a distributed continuous data transfer system (kafka), a distributed real-time computing platform (Storm) and finally a distributed storage system (hdfs).

Once the platform was established, I implemented a use case in it, consisting of a generation of data pretending to be wind measurements collected in two cities, transmitting the data to the system, but eliminating measurements greater than 50 m/s preserving and recording the trace and the number of measurements eliminated, calculating the latency at the time the data are stored in the local machine, and also storing this data in a system hdfs.

It is therefore established a flow of data from its generation to its storage, passing through a series of processes from which you can get information that can be included in the system.

Palabras clave (entre 4 y 8):

IoT, BigData, Kafka, Storm, HDFS, Eclipse, Java, maven

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Desde hace tiempo internet se ha convertido en algo básico en nuestras vidas, y cada más, nos estamos acostumbrando a escuchar el término “Internet of Things” (IoT) o Internet de las cosas en español. Este concepto se identifica con un sistema de dispositivos o máquinas, equipados con tecnologías que permiten la recopilación de datos para que esos dispositivos o máquinas puedan comunicarse entre sí.
Las cosas (Things) o máquinas o dispositivos son básicamente sensores o actuadores. Un sensor es un dispositivo que puede detectar acciones o estímulos externos y responder en consecuencia (https://pacomaroto.wordpress.com/about/introduccion-a-la-internet-de-las-cosas/). Los sensores más comunes pueden ser de temperatura, o incluso el receptor GPS de nuestros teléfonos móviles.

Cada vez se aplica en más campos, entre los que podemos encontrar [1] [2]:

- Transporte/Logística, permitiendo mejorar la cadena de suministro debido a la información que aporta acerca de la situación y localización en la que se encuentran las mercancías en cada momento, así como todos los aspectos alrededor del coche conectado. En este sentido, las grandes compañías que controlan este mercado ya están hablando de la siguiente generación de coches conectados, que empezó con el coche sin conductor de google.
- Casas Inteligentes, cuyo valor radica en la optimización de los consumos de recursos en el hogar, aumentando el confort y la seguridad de los habitantes del mismo. La creación de empresas que ofrecen este tipo de servicios a los ciudadanos está creciendo considerablemente.
- Ciudades Inteligentes, dentro de las cuales se ubicarían las casas inteligentes. Tanto unas como otras tienen como objetivo aumentar la calidad de vida de las personas que las habitan. Los aparcamientos inteligentes, la gestión inteligente del tráfico o del alumbrado, son aspectos que facilitan la vida dentro de las ciudades y permiten disponer de más tiempo de calidad a sus ciudadanos, además de hacerlas más habitables.
- Fábrica Inteligente, con la que las empresas podrán tener un mayor control y seguimiento de toda la cadena de producción, y del ciclo de vida completo de sus productos. Actualmente esto se consigue identificando a los productos mediante etiquetas RFID, o integrando sensores a dichos productos desde los que emitir datos acerca de su estado. La producción se ve mejorada y se facilita el mantenimiento de la planta, permitiendo monitorizar las máquinas de forma remota y en tiempo real.
- Pequeño comercio, que podrá beneficiarse de la red, para anunciar sus productos dando importancia a la calidad basada en la proximidad al

cliente, además de tener la posibilidad de asociarse con otros comerciantes dentro de la misma oferta, repartiendo la gestión y llegando a más clientes.

- eSalud (eHealth), aportando como un gran beneficio la posibilidad de un seguimiento médico de forma remota, más aún cuando cada vez es mayor la cantidad de hogares unipersonales en el grupo de personas mayores.
- Smart Energy/Smart Grid, que puede verse como un ámbito solapado o coincidente con las casas y ciudades inteligentes, ya que su principal objetivo es el ahorro de energía. Medición inteligente, monitorización, control y comunicación inteligente para conseguir múltiples beneficios (detección rápida de interrupciones de suministro, mayor control del consumo, mayor aprovechamiento de las fuentes de energía, aumentar la eficiencia en el consumo en base al análisis del mismo, etc…)
- Wearables (ropas/accesorios conectados), comenzando por los relojes inteligentes o smartwatches que están tan de moda, como las pulseras de actividad en el control del deporte y del deportista, hasta pulseras identificadores en hoteles y establecimientos.
- Granja inteligente o Smart farming, siendo este ámbito no tan claro como los anteriores aunque con un gran potencial para su desarrollo. La gestión de la explotación ganadera o agrícola puede ser gestionado de forma similar a una fábrica, controlando los animales o los cultivos, su distribución y su venta.

La IoT por lo tanto, pone en contacto diferentes elementos de una red, que hasta el momento quizá no se habían visto como tal. La relación que tienen esos elementos una vez que están en contacto, cómo acceden o comparten la información necesaria, cómo la procesan, qué provecho sacan de dicha información, entra dentro del ámbito del Big Data, y del Big Data Analytics.

La relación entre estos dos ámbitos tecnológicos se aprecia claramente.

Big Data es un término que hace referencia a grandes cantidades de datos, tanto estructurados como no estructurados [3].

Cada vez existen más datos circulando por la red, la información es muy importante actualmente, y el futuro de las tecnologías tiene que dar solución a la gestión de estos datos y al análisis de los mismos.

Podría decirse que el objetivo del Big data consiste en utilizar estos datos para aprender sobre patrones y tendencias que pueden ser usados para mejorar nuestra salud, transporte, energía, conservación y estilo de vida [4].

Las herramientas para el análisis de datos, pueden manejar gran cantidad de ellos que son transmitidos desde dispositivos pertenecientes a sistemas IoT, que producen un flujo continuo de información.

Según va creciendo la Internet de las cosas, más se demanda por parte de las empresas, las capacidades de Big Data [5].

Las organizaciones son conscientes de que quieren usar IoT y BigData, pero tienen que encontrar la forma y las herramientas adecuadas para hacerlo.

Hay muchos ejemplos de empresas que lo van consiguiendo. En la industria del transporte, se están colocando sensores en los vehículos como forma de realizar un seguimiento de sus rutas, proporcionando información sobre la situación de dichos vehículos en cada momento y permitiendo optimizar las rutas y por lo tanto mejorar su productividad, en base al análisis de dichos datos.

Muchos centros dedicados a la investigación, están trabajando en estos ámbitos en los que se avanza de forma conjunta. En mi entorno laboral actualmente existen áreas dedicadas a estos ámbitos, y es muy interesante la forma en la que crear sinergias entre ambos grupos,

Me resulta interesante experimentar, en una forma controlada, sobre cómo integrar estos dos mundos y hacerlos funcionar, desde cero, y en un entorno aislado, simulando entradas que hagan que todo el sistema se ponga a trabajar y ofrezca los resultados deseados. Conocer los problemas y los conceptos, desde un enfoque local, para saber extrapolar esas soluciones y conocimientos adquiridos, a sistemas más complejos y distribuidos. Esta es la base de mi motivación para el desarrollo del trabajo que expongo en los puntos que vienen a continuación.

1.2 Objetivos del Trabajo

Considero como principal objetivo, la integración de un sistema distribuido de transferencia continua de datos (Kafka), con una plataforma de computación en tiempo real también distribuido (Storm), ambos de Apache. Esta plataforma resultante es la unión entre los producers de datos (IoT) y los analizadores de información (BigData).
Con esa plataforma pretendo hacer un caso de uso en el que obtener datos de medición de la velocidad del viento en diferentes ciudades, transmitirlas al sistema, gestionar y realizar una serie de procesos con esos datos y obtener finalmente unos resultados que pudieran servir a los habitantes de esas ciudades para inferir algún tipo de acción asociado a ese conocimiento.

Como objetivos secundarios y que tienen que cumplirse para darse el principal, están el conocimiento de los sistemas involucrados, la gestión e implementación exitosa de ambos sistemas en un mismo entorno, la elección y prueba de un IDE de desarrollo con el que manejar los elementos softwares necesarios, y el desarrollo de un ejemplo, simulando en varios puntos del proceso los datos producidos por agentes externos al sistema completo.

1.3Planificación

Debido a mi interés en realizar un trabajo que se ajustara al ámbito en el que actualmente se está trabajando dentro de mi departamento, en mi lugar de trabajo, las dos primeras fases del trabajo fueron más largas de lo que creo que podrían ser en trabajos cuyo alcance y objetivos estaban definidos en el plan docente del TFM. No obstante, estas fases eran cruciales para el establecimiento correcto y realista del alcance del trabajo.
Para la realización exitosa del trabajo, el primer paso fue establecer un plan, junto con el director del mismo. Las fechas serían aproximadas y servirían para organizar el tiempo restante hasta la entrega del mismo.

En el siguiente diagrama se pueden apreciar las fases del mismo:

Ilustración 1: Diagrama Gantt organización del trabajo

FASE I: Planteamiento.
Esta fase corresponde con la elección de un planteamiento inicial del trabajo, que cumpliera con las expectativas de aprendizaje que me había propuesto, y con el alcance y los objetivos que se esperan de un trabajo de final de master.
El interés por mi parte se concentraba en adquirir conocimientos sobre temas relacionados con IoT, pudiendo incluir aspectos de CPS (Cyber Physical systems).
Básicamente se analizó cómo las tecnologías existentes podrían integrarse para dar soporte a aplicaciones IoT reales.

FASE II: Diseño alto nivel. Colaboraciones.

Una vez trabajado el alcance y el contexto del trabajo, analizamos diferentes opciones para desarrollar. Iván contactó con investigadores del CSIC y de la UPC para plantearles la situación y ver la manera en la que podríamos colaborar. Uno de los proyectos en los que estaban trabajando, estaba relacionado con mejorar las predicciones de velocidad del viento en entornos muy cambiantes (por ej. Delta del Ebro) mediante soluciones IoT en las que se podrían usar por ejemplo sensores instalados en teléfonos móviles de alguna comunidad de windsurf o similar. Se podrían obtener mejoras en la información mostrada debido a la retroalimentación en las predicciones, así como analizar la cantidad óptima de sensores necesarios para una mejora en las predicciones.
Planteamos centrar el trabajo en la adquisición de datos y su gestión, desde un dispositivo (por ejemplo un smartphone) hasta la cloud (podríamos analizar otras posibilidades como edge computing o entornos híbridos). Como primera

aproximación, estos datos serían creados ad-hoc, para no depender de una instalación física de sensores. Nos pareció buena opción para validar el modelo que se desarrollaría, dentro de un entorno manejable y predecible.

También valoramos la posibilidad de realizar este tipo de colaboración con un grupo de investigación en el campo de la oceanografía, al que Iván conocía.
Estas colaboraciones, aunque muy interesante, resultaron que requerían de un alcance, duración y dedicación que se quedaba muy fuera de lo que disponíamos para este trabajo, por lo que optamos por reformular el planteamiento, como una solución más genérica que se pudiera aplicar en esos casos de uso en el futuro.

En esta fase por lo tanto, fijamos llegar a un framework o plataforma general para poder aplicar diversos casos de uso.

FASE III: Diseño teórico plataforma.
En esta fase se diseñó la plataforma a desarrollar, en la que se integrarían las distintas tecnologías que iban a formar parte del esquema.
Esta fase se corresponde con los puntos 3.1, 3.2 y 3.3 de esta memoria.

FASE IV: Integración tecnologías. Ejemplo.
Fase en la que se integran todas las tecnologías en una plataforma y se valida dicha infraestructura mediante un ejemplo.
Esta fase se corresponde con el punto 3.4 de esta memoria.

FASE V: Implementación caso de uso.
En esta fase se realiza la implementación de un caso de uso sobre la plataforma creada.
Esta fase se corresponde con el punto 4 de esta memoria.

FASE VI: Revisión y conclusiones.
Finalmente se establece un conjunto de posibles mejoras y de pasos sugeridos para implementar a una mayor escala este sistema.
Se obtienen además una serie de conclusiones, y se documenta todo el trabajo en forma de memoria.
Esta fase se corresponde con los puntos 5 y 6 de esta memoria, y la presentación del trabajo.

1.4 Enfoque y método seguido

Para la consecución de los objetivos anteriormente descritos, me he propuesto desarrollar pequeños hitos que van a ser completados para conseguir hitos mayores y más complejos.

Los pasos seguidos son:

- Establecimiento teórico de los elementos de la plataforma a desarrollar. Esquema de sistema.
- Elección de SO

- Búsqueda de tutoriales para instalación de Kafka. Instalación y puesta en marcha.
- Búsqueda de tutoriales para instalación de Storm. Instalación y puesta en marcha.
- Integración mediante un ejemplo de los dos sistemas. Superación de problemas, incompatibilidades etc…
- Implementación del diseño teórico previo, en los elementos que componen la plataforma.
- Pruebas y ajustes.
- Establecimiento de posibles mejoras y de pasos para implementar a una mayor escala este sistema.

1.5 Breve descripción de los otros capítulos de la memoria

Esta memoria consta de las siguientes secciones:

- Conceptos. Tecnologías

Un estado del arte de las distintas tecnologías involucradas y de los conceptos más importantes de las mismas.

- Plataforma

Proceso de instalación de la infraestructura necesaria para la consecución de los objetivos del proyecto, y sobre la que se ha realizado todo el proceso, así como los pasos y avances que se han seguido para la integración de todas esas tecnologías en una infraestructura de soporte a la implantación de un ejemplo simple pero completo.

- Esquema IoT

Descripción del desarrollo del caso de uso sobre la plataforma instalada, estableciendo la correspondencia de cada elemento del sistema con su función dentro de ese caso de uso.
Cuáles han sido las decisiones que he ido tomando, las fases que he ido siguiendo y los resultados obtenidos.

- Siguientes pasos.

Descripción de los elementos de la plataforma resultado, enfocando este resumen como una serie de aspectos a cambiar o mejorar para construir o implementar un sistema "real" más grande para este mismo caso de uso.
Estudio de lo que sería un “Sistema objetivo teórico” y recomendaciones parar para escalarlo.

- Conclusiones.

Reflexiones acerca de las lecciones aprendidas y conocimiento adquirido durante todo el proceso de creación de este sistema de IoT usando BigData, en un entorno controlado y simulado, a pequeña escala.

2. Conceptos. Tecnologías.

Para la realización de este trabajo se han visto envueltas una serie de tecnologías cuyos conceptos básicos se exponen a continuación.

2.1 Kafka

Según la Wikipedia, Apache Kafka es una plataforma open-source para el procesado de datos en streaming o transmisión continua, realizada por la Apache Foundation e implementada en Scala y java [6].
Desde la propia página de Apache se indican como principales estas tres capacidades de Kafka [7]:

- Permite publicar y subscribirse a flujos continuos de información. Similar a una cola de mensajes o un sistema empresarial de mensajes.
- Permite almacenar flujos continuos de información de forma tolerante a fallos, o lo que es lo mismo, podrá seguir funcionando aún en caso de que alguno de sus elementos falle.
- Permite procesar flujos continuos de información en tiempo real, según van apareciendo.

El uso de esta plataforma proporciona los siguientes beneficios [8]:

- Confiabilidad: distribuido, particionado, replicado y tolerante a fallos.
- Escalabilidad: se puede escalar fácilmente sin necesidad de “apagarlo”
- Durabilidad: utiliza log distribuido de commit, lo que significa que los mensajes permanecen en el disco y pueden ser accedidos lo más rápido posible.
- Rendimiento: alto rendimiento para la publicación y suscripción de mensajes. Mantiene un rendimiento estable incluso si se almacenan grandes cantidades de mensajes.

Kafka es muy rápido por lo que garantiza cero tiempo de inactividad y cero pérdida de datos.

Debido a estas características, existen unos casos de uso para los que esta plataforma potencia su rendimiento. Estos casos de uso son:

- Monitorización de datos y métricas. Puede consistir en agrupar estadísticas provenientes de diferentes aplicaciones distribuidas, y alimentar con ello sistemas de datos centralizados.
- Agregación de registros. Recopilar registros de varios servicios y transformarlos a un formato estándar accesible para múltiples consumers.
- Procesamiento de secuencias: En conjunción con otros sistemas como Storm o Spark Streaming, para leer datos, procesarlos, y escribirlos de distinta forma para poder ser accedidos por diferentes usuarios y aplicaciones.

Los principales elementos del Sistema KAFKA son los que aparecen en la siguiente imagen:

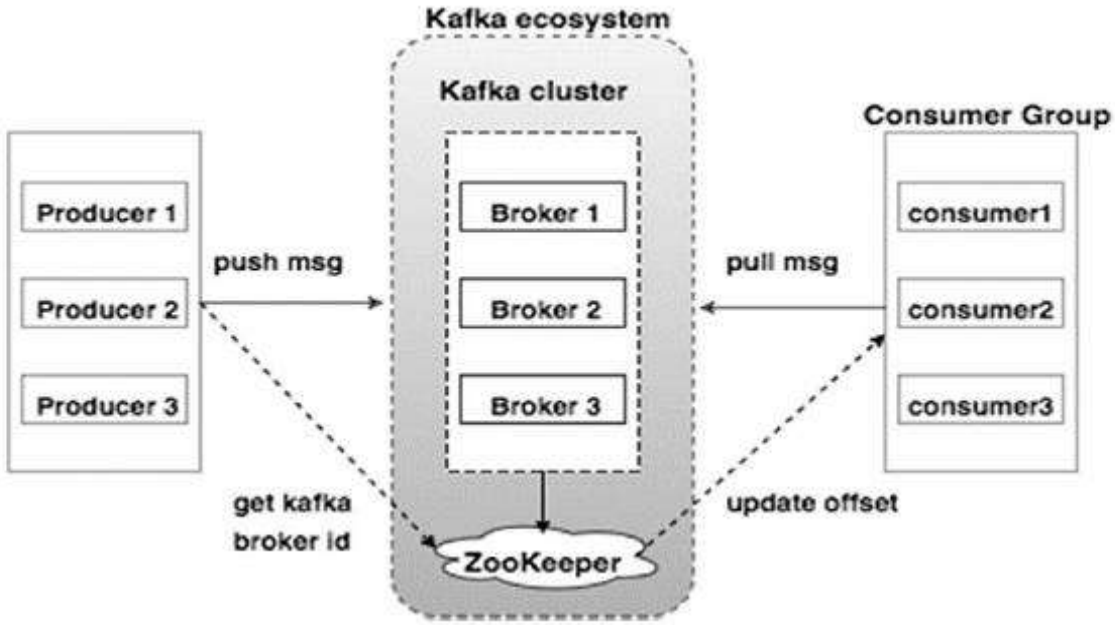

Ilustración 2: Kafka cluster architecture [9]

Producer: Los Producers inyectan datos (mensajes) a los brokers. Cuando se arranca un nuevo bróker, los producers lo detectan y automáticamente le envían un nuevo mensaje. Estos producers no esperan un acuse de recibo de los brokers por lo que envían mensajes tan rápido como dicho bróker los pueda gestionar.

Broker: El clúster de Kafka consiste normalmente en varios brokers para poder mantener el balanceo de carga. Estos brokers son stateless (no gestionan datos acerca de su estado), por lo que utilizan ZooKeeper para mantener su estado. Un broker puede manejar cientos de miles de lecturas y escrituras por segundo y cada broker puede manejar TB de mensajes sin que el rendimiento se vea afectado. La elección del broker líder la puede realizar Zookeeper.

ZooKeeper: Como hemos descrito en la descripción de Broker**,** ZooKeeper se utiliza para la gestión y coordinación de los mismos. El servicio de ZooKeeper se utiliza principalmente para notificar, tanto a los producers como a los consumers que existe un nuevo bróker en el sistema, o que alguno de ellos ha fallado. Entonces, el producer y el consumer, eligen a otro bróker del sistema y comienzan a trabajar con él.

Consumers: Como ya se ha comentado anteriormente, los brokers son "stateless", por lo que los consumers tienen que saber cuántos mensajes se han consumido usando el "partItion offset" (posición en la que está actualmente referente al mensaje que está consumiendo o leyendo). Si el consumer reconoce un determinado offset, esto quiere decir que este consumer ya ha consumido todos los mensajes anteriores. El consumer emite una solicitud de pull (extracción) asincrónica al broker para tener almacenados unos cuantos bytes listo para consumir. Los consumers pueden rebobinar o saltar a cualquier

punto de una partición simplemente suministrando un valor de offset. Este offset del consumer es notificado por ZooKeeper.

El funcionamiento general de Kafka consiste en ejecutarse como un cluster en uno o más servidores. Este cluster almacena los flujos de información o de registros en categorías llamadas “topic”. Cada uno de estos registros consta de una clave, un valor y un timestamp o a marca de tiempo.

Pero el elemento clave en Kafka es el **topic**.
Los producers publican sus registros en un topic para que los consumers se subscriban a ellos.
Los topics se dividen en particiones. Para cada topic, Kafka mantienen un log de particiones, de esta forma:

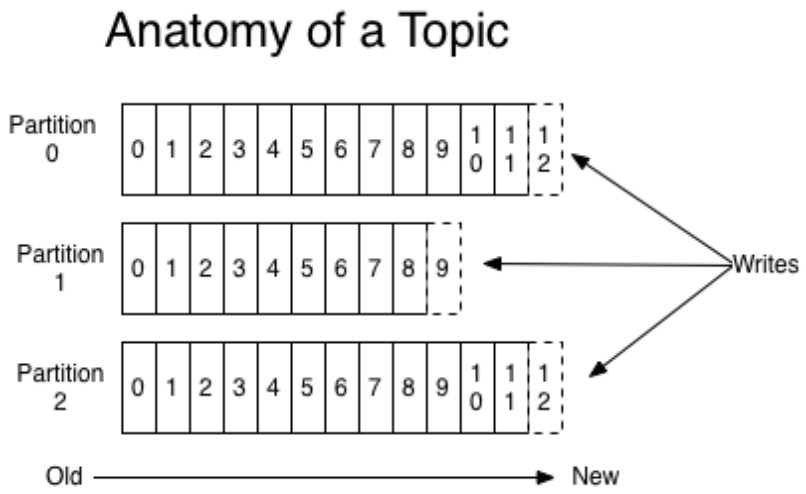

Ilustración 3: Topic [10]

Cada partición es una secuencia ordenada e inmutable de registros que se agrega continuamente a un log de commit estructurado. A los registros de las particiones se les asigna un número de identificación secuencial llamado offset que identifica de manera única cada registro dentro de la partición.

El clúster de Kafka conserva todos los registros publicados, sean o no consumidos, utilizando un “período de retención” configurable. Una vez que dicho periodo ha terminado, los registros ya no estarán disponibles para su consumo.

Los consumers solamente gestionan los metadatos correspondientes al offset o posición en el registro. Este offset avanza de forma lineal a medida que lee registros, aunque en realidad podría consumir dichos registros en el orden que quisiera, debido a que almacena la posición.

El flujo dentro del sistema siguiendo un esquema de public-subscribe sería el siguiente:

Un producer envía un mensaje a un topic, cada x tiempo.
Broker almacena los mensajes en las particiones configuradas para ese topic en particular. Se asegura de que los mensajes sean compartidos por igual entre las particiones. Si el producer envía dos mensajes y hay dos particiones, Kafka almacenará un mensaje en la primera partición y el segundo mensaje en la segunda partición.
Los consumers se subscriben a un topic específico.
Una vez que el consumer se suscribe a un tema, Kafka proporcionará la posición (offset) actual del topic al consumer y guardará además ese offset en Zookeeper.
Consumers pedirán nuevos mensajes a Kafka, a intervalos regulares de tiempo.
Cuando el sistema Kafka reciba nuevos mensajes de los producers, los enviará a los consumers.
Los consumers reciben el mensaje y lo procesan.
Una vez procesado el mensaje por los consumers, estos envían un “acuse de recibo” al bróker.
Cuando Kafka recibe el acuse de recibo, cambia el offset al nuevo valor y lo actualiza en Zookeeper.
Este flujo parará cuando el consumer deje de realizar peticiones de mensajes.

Recapitulando, cuando se trabaja en ambientes relativos a la tecnología BigData, se usan cantidades enormes de data, situación en la que utilizaremos el sistema Storm, cuya descripción se muestra en el siguiente punto.

2.2 Storm

Apache Storm es un sistema open Source de computación en tiempo real distribuido y libre. Storm facilita el procesamiento de forma fiable de flujos de datos en cantidades sin límite, es simple, y se puede utilizar con cualquier lenguaje de programación [11].

Storm es fácilmente integrable con las tecnologías de colas y bases de datos más utilizadas. Una topología Storm consume flujos de datos y procesa esos flujos arbitrariamente de formas complejas, reparticionando los flujos entre cada etapa del cálculo, según se necesite.

Los beneficios que ofrece Apache Storm son muchos, entre los que se encuentran:

- Es open Source, robusto, robusto y fácil de usar. Puede ser usado tanto en grandes como pequeñas empresas:
- Es tolerante a fallos, flexible, confiable y soporta cualquier lenguaje de programación.
- Permite procesamiento de flujos de datos en tiempo real.
- Es muy rápido debido a su gran capacidad de procesamiento de datos.
- Puede mantenerse en ejecución incluso bajo una carga creciente, ya que va añadiendo recursos según lo va necesitando. Es altamente escalable.

- Realiza actualización de datos y entrega extremo a extremo en segundos o minutos dependiendo del problema. Tiene latencia muy baja.
- Proporciona inteligencia operativa (OI: una forma de análisis dinámico de los procesos, en tiempo real).
- Proporciona un procesamiento de datos garantizado incluso si se pierden los mensajes o alguno de los nodos conectados en el clúster.

Apache Storm lee flujos de datos sin procesar desde un extremo, y los pasa a través de una secuencia de pequeñas unidades de proceso, al otro extremo del ciclo.

Existen dos clases de nodos en un cluster de Storm, el master y los workers. El nodo master lanza un demonio (proceso que se ejecuta en un segundo plano) llamado “Nimbus”, responsable de distribuir código por el cluster, asignar tareas a las máquinas y monitorizar el funcionamiento en busca de fallos.
Cada nodo worker ejecuta un demonio llamado “Supervisor”, según el trabajo asignado a su máquina, inicia y detiene los procesos según sea necesario, basándose en las asignaciones de Nimbus. Cada proceso del worker ejecuta un subconjunto de una topología; Una topología en ejecución consiste en muchos procesos de worker extendidos a través de muchas máquinas.
Un esquema de este cluster sería el siguiente:

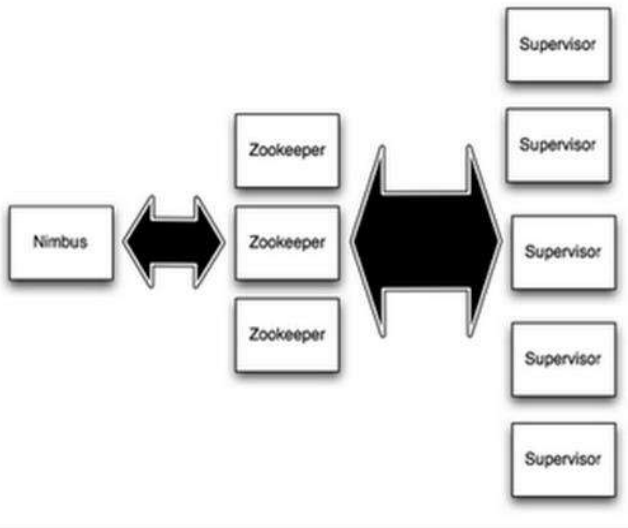

Ilustración 4: Cluster Storm [12]

La coordinación entre Nimbus y los Supervisors se hace a través de Zookeeper, ya que los nodos descritos son stateless.

En cuanto a la aplicación de Storm para computación en tiempo real, nos encontramos con el concepto de “Topologías” que se despliegan en un cluster. Una topología es un grafo de computación, donde cada nodo contiene lógica de proceso y donde los enlaces entre los nodos indican cómo pueden ser transmitidos los datos entre los nodos que forman esa topología.

El diagrama que ilustra los principales conceptos que se manejan en una topología Storm es el siguiente:

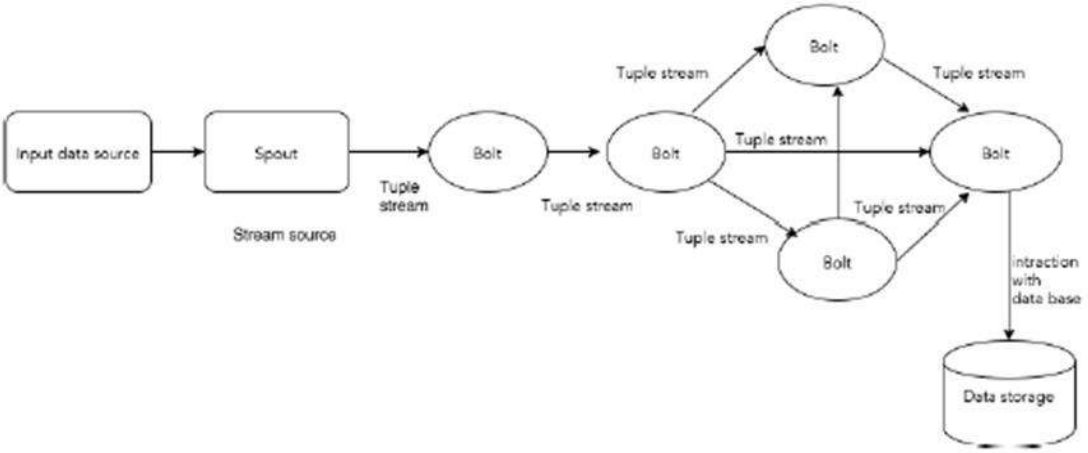

Ilustración 5: Storm Topology [13]

Una Tupla (tuple) es la principal estructura de datos en Storm. Es una lista ordenada de elementos, de cualquier tipo. Generalmente tiene la forma de un conjunto de valores separados por comas.

Stream es una secuencia no ordenada de tuplas.

Spout es la fuente de “Streams”. Storm acepta datos de entrada de fuentes de datos sin procesar como Twitter Streaming API, Apache Kafka queue, Kestrel queue, etc.

También se pueden construir spouts que lean datos almacenados. ISpout es la interface que proporciona Apache para implementar los Spouts necesarios en un sistema. Existen además otros interfaces más específicos como IRichSpout, BaseRichSpout, KafkaSpout, etc.

Bolts son las unidades de procesamiento. Pasan datos a otros bolts para que los procesen y emitan un nuevo stream como salida. Pueden realizar labores de filtrado, agregación, unión e interacción con datos almacenados. Pueden llegar a realizar procesados similares al mapReduce de Hadoop. IBolt es el principal interface que proporciona la plataforma para implementar bolts. Existen también interfaces específicos como IRichBolt, IBasicBolt, etc.

Para definir una topología, uno de los aspectos que hay que definir es la especificación para cada bolt de qué stream va a recibir como input. Esto se define mediante la característica de “groupping” y define cómo se va a particionar ese stream entre las tareas del bolt.

Puede ser de ocho formas distintas, aunque se pueden definir a medida más si fueran necesarias.

- **Shuffle grouping**: Tuplas se distribuyen de forma arbitraria entre las tareas del bolt.
- **Fields grouping**: Particionadas por un campo especificado.
- **Partial Key grouping**: Particionadas por un campo especificado, pero con un balanceado de carga entre dos bolts
- **All grouping**: Se replican a través de todas las tareas del bolt.
- **Global grouping**: El Stream por completo va a una de las tareas del bolt, a la que tiene ID menor.
- **None grouping**: No importa cómo se agrupan. Equivalente a Shuffle.
- **Direct grouping**: El producer de la tuple decide a qué tarea del consumer va a dirigirse.
- **Local or shuffle grouping**: Si el bolt objetivo tiene una o más tareas en el mismo proceso de worker, las tuplas irán a las que estén en proceso, en otro caso será un shuffle groupping normal.

Una topología está en ejecución hasta que se pare por el usuario. Storm reasignará automáticamente cualquier tarea que haya fallado. Además Storm garantiza que no se perderán datos, aun cuando las máquinas se caigan y los mensajes desaparecen.

2.3 Hadoop HDFS

Apache Hadoop es un entorno de trabajo open-source que se utiliza para almacenaje distribuido y procesamiento de conjuntos de datos de BigData, usando para ello el modelo de programación MapReduce [14].

La parte básica de Apache Hadoop consiste en una parte de almacenaje, conocida como HDFS (Hadoop Distributed Flie System) y otra parte de procesamiento que trabaja con el modelo MapReduce.

Hadoops divide los ficheros en grandes bloques y los distribuye a través de los distintos nodos que forman un cluster. A continuación, transfiere el código empaquetado a los nodos para que sean procesados en paralelo.

HDFS es altamente tolerable a fallos y está diseñado para que sea desplegado en hardware de bajo coste. Proporciona además alto rendimiento en el acceso a los datos y es adecuado para aplicaciones que tienen grandes conjuntos de datos.
HDFS actualmente es un subproyecto Apache Hadoop.

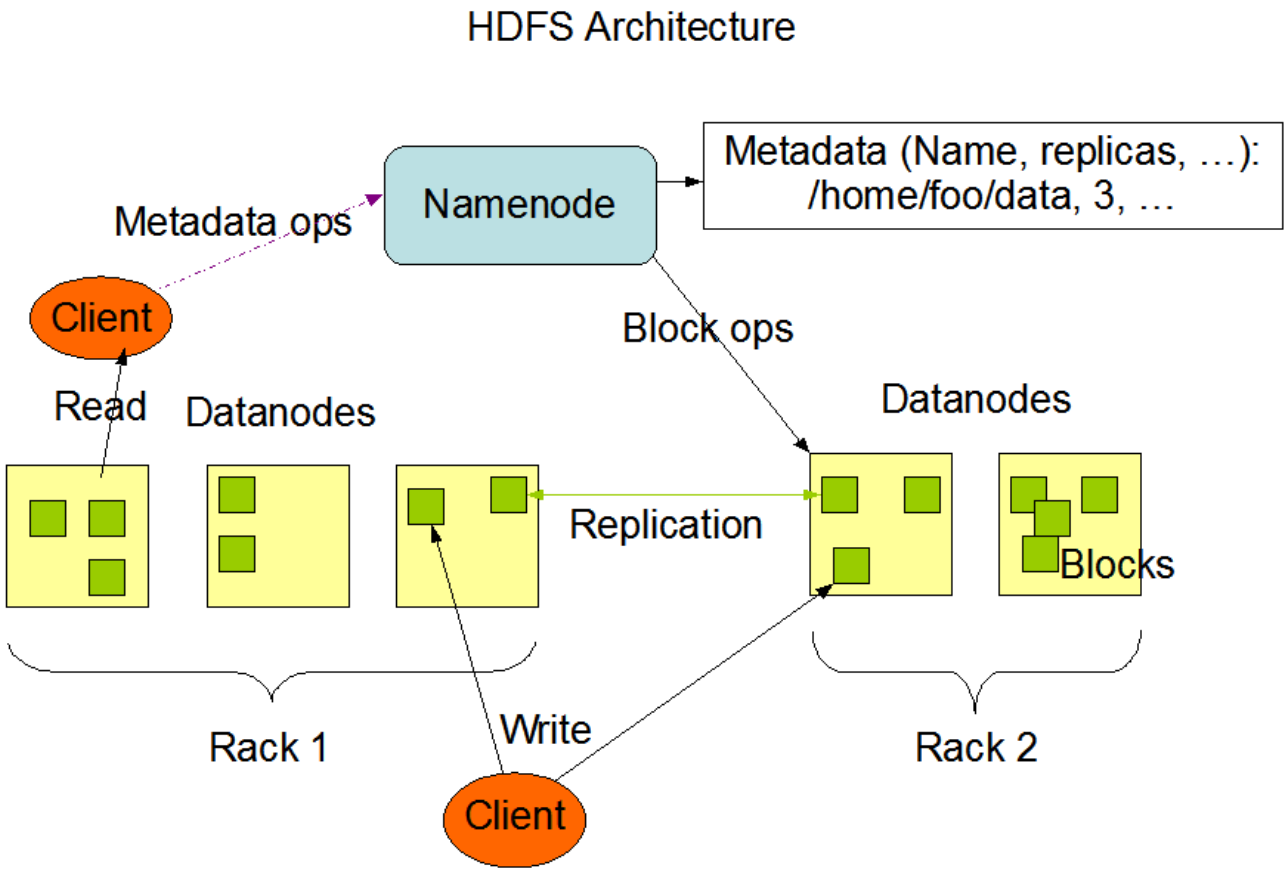

Ilustración 6: HDFS arquitectura [15]

HDFS tiene una arquitectura maestro esclavo. Un clúster HDFS consta de un único NameNode, que es un servidor maestro que gestiona el espacio de nombres del sistema de archivos y regula el acceso a los archivos por parte de los clientes. Además, hay un número de DataNodes, por lo general uno por nodo en el clúster, que gestionan el almacenamiento asociado a los nodos en los que se ejecutan. HDFS expone un espacio de nombres de sistema de archivos y permite almacenar datos de usuario en archivos. Internamente, un archivo se divide en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes. El elemento NameNode ejecuta “namespace” (conjunto de nombres en el que todos los nombres son únicos) operaciones del sistema de archivos como abrir, cerrar y renombrar archivos y directorios. También determina el mapeo de bloques a DataNodes. Los DataNodes son responsables de atender las solicitudes de lectura y escritura de los clientes del sistema de ficheros. Los DataNodes también realizan operaciones como creación, eliminación y replicación de bloques a partir de instrucciones del NameNode.

3. Plataforma

El objetivo planteado para este trabajo, tiene como requisito montar una infraestructura en la que poder desplegar todos los sistemas necesarios, arrancar los servicios que van a verse envueltos en el sistema global, realizar

pruebas, obtener resultados y poder instalar de forma incremental, los diferentes elementos de mi plataforma.
Este trabajo comienza con una base, sobre ella se instala el sistema Kafka, que una vez probado y funcionando, requiere de una integración con Storm. Esta integración es compleja y requiere de mucho esfuerzo en cuanto a pruebas y soluciones de los problemas encontrados.
En los puntos de los que consta este apartado, describo la tecnología que finalmente me ha sido de utilidad para la consecución de los objetivos, así como los problemas que he ido encontrando en el camino.

3.1 Requisitos de base.

Como primer paso para implementar el sistema de Iot y Big data planteado, necesito tener una plataforma activa y funcional sobre la que trabajar.
Esta plataforma en un primer momento, consiste en la instalación de Kafka, prueba de funcionamiento, instalación de Storm, prueba de funcionamiento y una vez comprobado el buen funcionamiento de ambos sistemas, integrarlos para que sea posible la comunicación entre ellos.

El sistema operativo existente en mi ordenador es windows 10, pero por lo que leo en páginas y tutoriales, lo mejor para este tipo de plataformas es Linux, por lo que opto por desarrollar todo el sistema en una imagen virtual, con sistema operativo Ubuntu.
Creo que además me proporciona independencia respecto al software que pueda haber en mi máquina y me permite hacer copias de todo el sistema completo para tener copias de seguridad a las que poder volver cada vez que hago una modificación importante en el sistema.

Queda decidido entonces que voy a desarrollar toda mi infraestructura sobre:

- Máquina virtual (VirtualBox) con imagen Ubuntu 12.04

El software de VirtualBox está disponible en su página https://www.virtualbox.org/wiki/Downloads, y la imagen con el sistema operativo la he obtenido de una página que ofrece de forma libre, imágenes de distintos sistemas operativos y para distintos softwares de máquinas virtuales. Me refiero a osboxes.org, de donde he obtenido una imagen con Ubuntu 12.04. No he elegido la versión más moderna porque muchas veces las herramientas no están actualizadas para los sistemas más recientes. De esta forma evito problemas debidos a este tipo de desactualizaciones.

La siguiente instalación que necesito hacer es la de java, ya que tanto Kafka como Storm necesitan java para funcionar. Es uno de los requisitos que me encuentro en los dos tutoriales que estoy siguiendo, tanto el publicado en https://www.tutorialspoint.com/apache_kafka/apache_kafka_installation_steps.htm como en el propio de Apache https://kafka.apache.org/quickstart

Instalo java Java SE Development Kit 8, ya que voy a necesitar implementar código para construir aplicaciones:

- Java 1.8.0_121 jdk (de la página de Oracle, aunque actualmente haya versiones más nuevas)

Una vez que he instalado java, configuro la variable JAVA_HOME, para añadirla a la variable PATH del sistema. De esta forma me aseguro que mi sistema siempre sabrá donde encontrar la instalación de java, siempre que la necesite.
Esta configuración se realiza en el fichero ~/.bashrc del sistema, añadiendo estas líneas al contenido del mismo:

```
export JAVA_HOME =/usr/lib/jvm/java-8-oracle
export PATH=$PATH:$JAVA_HOME/bin
```

Después de instalar java, se puede comprobar que se ha instalado esta versión correctamente con el siguiente comando:

- java –version

```
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)`
Java HotSpot(TM) Client VM (build 25.121-b13, mixed mode)
```

Después de sufrir unos cuantos problemas, debidos a que no se instalaban correctamente algunas herramientas, no sólo con esta instalación de java sino con las siguientes instalaciones, he llegado a la conclusión de que es una buena práctica reiniciar el sistema cada vez que se realiza cualquier instalación y se quiere realizar otra instalación a continuación. No siempre es estrictamente necesario, pero evita errores debidos a configuraciones o variables que se confirman o toman valor global en el arranque del sistema operativo.

Como herramientas base que no pertenecen a la plataforma integrada, también he necesitado el IDE de eclipse para desarrollar aplicaciones en java, y maven como herramienta para la gestión y construcción de proyectos Java, que se integra en eclipse. Estas dos tecnologías las uso más adelante, para la integración de las dos plataformas, por lo que los detalles de configuración y cualquier otro detalle referente a dicha integración, estarán incluidos en el punto correspondiente del proceso.

3.2 Kafka

Una vez que dispongo de una máquina virtual donde tengo un SO Ubuntu y java 8 instalado, comienzo con la instalación de Kafka.
Las instrucciones por las que me voy guiando son las que aparecen en la página de Apache respecto a Kafka, en la sección quickstart (https://kafka.apache.org/quickstart) y mantengo como tutorial de contraste el proporcionado por tutorialspoint, en la sección de installation steps (https://www.tutorialspoint.com/apache_kafka/apache_kafka_installation_steps.htm).

La primera decisión que tengo que tomar hace referencia a la versión de kafka que voy a instalar. Mientras que en Apache se indica la versión kafka_2.11-0.10.2.0, en tutorialspoint en cambio optan por la versión kafka_2.11_0.9.0.0.
En un primer intento, opto por instalar la versión indicada en la página de Apache y seguir las instrucciones a partir de ahí. El funcionamiento es correcto, puedo arrancar, emitir y recibir mensajes desde y por Kafka, pero de forma aislada. El problema viene más adelante a la hora de integrar las dos plataformas e intentar un funcionamiento conjunto. Si no instalo dos versiones compatibles, puedo tener muchos errores en ejecución que pueden ser difíciles de arreglar o imposibles por incompatibilidad de dichas versiones.
Las versiones compatibles no son las más recientes. Voy a integrar Kafka versión 2.11-0.8.2.2 y storm versión 0.9.5, tal como se indica en el ejemplo de integración de tutorialsppoint, aunque las versiones actuales son de Kafka la versión 2.12-0.10.2.0 y de storm la versión 1.0.3. Pero estas dos versiones de momento no son compatibles para su integración. Esto no se indica en el tutorial, y se puede pensar que esas versiones eran las que estaban recientes en el momento que se realizó dicho tutorial. He comprobado que no, que la integración no funciona para versiones más recientes de ninguna de las dos plataformas.
Aunque realicé la instalación y las pruebas con una versión que no es la que finalmente forma parte de mi sistema global, los pasos a seguir son equivalentes independientemente de la versión que se instale, y para mayor claridad, indico la versión que finalmente ha resultado ser la correcta.

Por otra parte, y aunque en tutorialspoint se indica una instalación independiente de zookeeper, que en un principio realicé, Kafka lleva un zookeeper por defecto en su propia instalación y no hace falta esa instalación separada. Esta decisión me evita tener que configurar enlaces externos a la plataforma de Kafka que viene por defecto, y así me aseguro que se usará ese zookeeper de forma predeterminada.
No obstante, y ya que desde storm también se hace uso de zookeeper, habrá que indicar el modo de acceso y algunos parámetros para su uso desde storm.

A partir de ahora por tanto, arrancaré una instancia de zookeeper existente en la propia instalación de Kafka.

Instalo por tanto, la versión de Kafka 2.11-0.8.2.2, en una carpeta donde voy a incluir ambas instalaciones y todo mi entorno a partir de ahora: Kafka-storm.
Una vez descargado y descomprimido el fichero con la instalación de Kafka, en Kafka-storm/kafka_2.11-0.8.2.2, abro un terminal, accedo a esa carpeta y escribo la siguiente línea de comandos:

bin/zookeeper-server-start.sh config/zookeeper.properties

Con ese comando estoy arrancando el servidor zookeeper, con la configuración existente en zookeeper.properties. Aquí, por defecto vienen los parámetros necesarios, siendo el más importante de ellos clientPort=2181.
No modifico el fichero de propiedades ya que por defecto tiene unos valores que funcionan en los casos generales.

Una vez arrancado zookeeper, el siguiente paso es arrancar el servidor de Kafka:

bin/kafka-server-start.sh config/server.properties

El fichero server.properties contiene una configuración para correr Kafka como single bróker. Si se desea arrancar un cluster multi-broker, hay que lanzar varios servidores, teniendo como fichero de propiedades cada uno el suyo particular. Estos ficheros de propiedades deben de modificarse para que recojan esa diferencia de brokers, de la siguiente manera:

Crear en la misma carpeta, tantos ficheros properties como brokers se quieran incluir en el cluster. Para cada uno de ellos (lo nombraremos como server-x.properties). Estos ficheros properties se modificarán incluyendo las siguientes líneas:

broker.id=x
listeners=PLAINTEXT://:9093 → incrementándose por cada broker adicional
log.dir=/…/kafka-logs-x

Esta configuración es una modificación a la del bróker por defecto que ya hemos lanzado y que consiste en:

broker.id=0
listeners=PLAINTEXT://:9092
log.dir=/…/kafka-logs

La forma de funcionar con un cluster de varios brokers se explica claramente tanto en el tutorial de tutorialspoints como en la página de apache. En este apartado nos concentraremos en indicar las acciones generales realizadas para tener la plataforma instalada y corriendo. No obstante, todas las comprobaciones y pasos de los tutoriales las he realizado con el objetivo de probar el buen rendimiento de la instalación e ir afianzando el conocimiento de cada una de las tecnologías involucradas.

Volvemos por lo tanto al sistema en el que hemos dejado corriendo un server zookeeper y un server Kafka.
Lo siguiente que se debe hacer es crear un topic sobre el que construir los mensajes. En este caso, el topic que voy a crear va a llamarse prueba. Abrimos un nuevo terminal y escribo el comando que lo crea que es el siguiente:

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic prueba

A modo de comprobación, y aunque con el comando anterior ya se puede ver el mensaje de “topic created” que aparece al terminar de crearlo, puede ser conveniente en ocasiones revisar los topics que ya están creados. Sobre todo teniendo en cuenta que resulta complicado destruir los topics creados ya que Kafka en esta versión solamente los marca para borrar. No es hasta versiones más recientes que Kafka permite borrar topics. De momento, para pruebas, una

vez parados todos los servers, borro a mano las carpetas con los datos de los topics creados, y una vez se apaga la máquina ya no queda constancia de dichos topics.

En el mismo terminal escribo:

bin/kafka-topics.sh --list --zookeeper localhost:2181

Y aparece una lista de los topics existentes, en este caso únicamente el que acabo de crear.

Ahora ya sólo falta emitir mensajes y leerlos. Para emitirlos, podemos hacer uso del cliente producer que viene por defecto con la instalación:

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic prueba

Y en este momento el terminal se queda a la espera de que se escriban por pantalla los mensajes que se quieren emitir. Cada mensaje se termina con un retorno de carro.
Escribo:

> Esto es un mensaje
> Esto es un segundo mensaje

En este punto he usado el producer que viene por defecto con la aplicación de Kafka, pero se puede hacer un API que use la clase kafkaProducer class, con lo que se pueden producir los mensajes que se quieran, con un determinado proceso que se necesite. Un API producer de muestra se ofrece en el tutorial de tutorialspoints, cuyo funcionamiento he comprobado y ejecutado correctamente.

Estos mensajes tienen que ser recibidos por un consumidor que necesite esa información. Por lo que se hace también uso del cliente consumer que viene con la instalación de Kafka, de la siguiente manera:

Para lanzar el producer hay que indicar el puerto 9092, y para lanzar el consumer se indica el 2181. Esto es así con un consumer propio de Kafka (consumer por defecto o con API creado específicamente) o si se quieren obtener los datos desde otro sistema, como es el caso de storm.

Abro otro terminal y escribo:

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic prueba --from-beginning

Y por el terminal aparece lo siguiente:

> Esto es un mensaje
> Esto es un segundo mensaje

Con esto queda patente que Kafka está correctamente instalado y funciona correctamente, en su configuración básica.

3.3 Storm

Una vez que Kafka está instalado y lo puedo lanzar las veces que quiera sin errores, procedo entonces a la instalación de storm 0.9.5, después de haber descargado el fichero y descomprimido en la carpeta que he creado anteriormente Kafka-storm/ apache-storm-0.9.5

En tutorialspoint, que es el tutorial que sigo, aparecen como primeros pasos, la instalación de java y la instalación de zookeeper, pasos que no realizo por tenerlos ya en el sistema.
Por lo tanto, lo primero a realizar es adaptar la configuración. El fichero que contiene dicha configuración es conf/storm.yaml, y las modificaciones a realizar consisten en incluir las siguientes líneas:

```
storm.zookeeper.servers:
- "localhost"
storm.local.dir: “/home/osboxes/kafka-storm/apache-storm-0.9.5/lib”
nimbus.host: "localhost"
supervisor.slots.ports:
- 6700
- 6701
- 6702
- 6703
```

Una vez modificada la configuración, y habiendo lanzado previamente el servidor zookeeper, me posiciono en la carpeta de instalación de storm y arranco el elemento nimbus.

Abro un terminal y escribo:

bin/storm nimbus

Abro otro terminal para arrancar también el elemento supervisor:

bin/storm supervisor

Y una vez que tengo arrancados los elementos principales de Storm, lanzo la ui, o user interface, abriendo otro terminal y escribiendo:

bin/storm ui

En este momento en el que ya están todos los elementos arrancados, abro un navegador y escribo la URL http://localhost:8080 para que aparezca una página con datos acerca del cluster de Storm y de la topología construida:

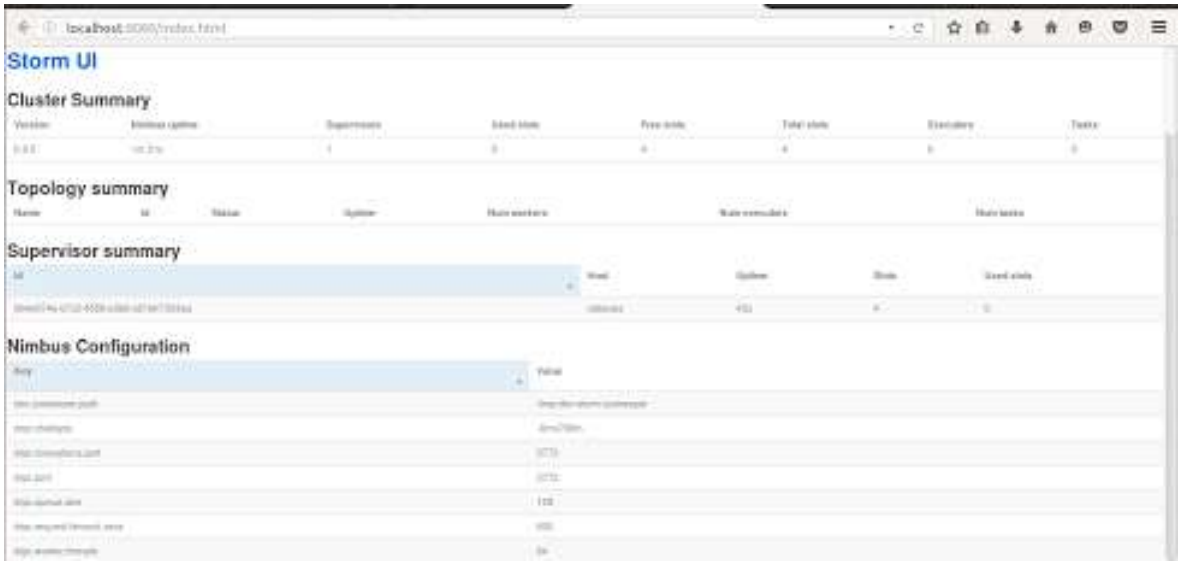

Ilustración 7: Storm UI

En esta página que aparece se pueden ver una serie de secciones con información del cluster y la topología [16].

Cluster Summary:

- Version: Version de Storm
- Nimbus up-time: Tiempo transcurrido desde que la instancia de Nimbus se ha lanzado
- Supervisors: Número de nodos en el cluster actualmente
- Total slots: Número de workers en el cluster.
- Used slots: Número de workers ocupados.
- Free slots: Número de workers libres
- Tasks: Una tarea es una instancia de Spout o de Bolt, por lo que esta propiedad sería la suma de las tareas lanzadas.
- Executors: Número de hilos.

Topology summary:

- Name: Nombre de la topología asignada cuando fue lanzada.
- Id: Un único ID que se le asigna a la topología cada vez que se lanza.
- Status: Estado que puede ser ACTIVE, INACTIVE, KILLED, o REBALANCING.
- Uptime: Tiempo que lleva lanzada la topología.
- Num workers: Número de workers usados en esta topología.
- Num executors: Número de executors usados en esta topología.
- Num tasks: Número de tareas usadas en esta topología.

Supervisor summary:

- Id – A unique identifier given to a Supervisor when it joins the cluster.
- Host – The hostname reported by the remote host. (Note that this hostname is not the result of a reverse look-up at the Nimbus node.)

- Up-time – The length of time a Supervisor has been registered to the cluster.
- Slots – Number of workers in the subject host. Normally storm evenly distributes the workers among all the hosts (nodes)
- Used slots – Number of workers that are occupied in the subject host

Nimbus Configuration:
Una serie de propiedades relativas a la configuración del Nimbus, como.

Tengo por tanto instalado Storm y funcionando correctamente.

3.4 Integración. Ejemplo.

Tanto Kafka como Storm, están instalados en mi sistema y funcionan correctamente de forma aislada, cada uno por separado.
Una vez que tengo preparada la plataforma, me dispongo a realizar una prueba de integración para comprobar que ambos sistemas están correctamente configurados para intercambiarse datos entre sí.

Para ello, y siguiendo los pasos de tutorialspoint voy creando los ficheros java necesarios tal y como están allí descritos:

- SplitBolt.java: clase de java que implementa la lógica para separar una frase en las palabras que la forman.
- CountBolt.java: clase de java que implementa la lógica para identificar palabras que son distintas a otras y que cuenta el número de ocurrencias de cada una de ellas.
- KafkaStormSample.java: clase de java que es un ejemplo en el que se crea una topología, se crea un bolt, se configura el spout y se crea la topología necesaria para leer los mensajes asociados al topic “test-topic”

En resumen, este ejemplo de integración, recoge las frases producidas por un producer de Kafka, las introduce en el sistema storm donde irá refinando los procesos hasta obtener un resultado. Como primer refinamiento, SplitBolt separa esas frases producidas pro Kafka, en palabras sueltas, para más tarde y como segundo refinamiento procesar esas palabras sueltas y contar las veces que una misma palabra ha aparecido en las frases producidas por Kafka. Este segundo refinamiento lo realiza el elemento CountBoult. Para que todo este proceso se lleve a cabo, se lanza KafkaStormSample que crea el cluster y monta una topología sobre él, indicando qué elementos van primero, cuáles reciben input de cuáles, y definirá el resultado del ejemplo completo.

Para este ejemplo existen dependencias internas con curator-client-2.9.1.jar y curator-framework-2.9.1.jar.
En un principio los ejemplos de API producer y consumer que he construido mientras probaba la instalación de Kafka, los he hecho directamente en un editor de texto y compilado y ejecutado por línea de comandas lanzando los ejecutables de java para ello. Esto es algo bastante incómodo e inefectivo, teniendo en cuenta que las dependencias pueden ser grandes y que el debug del software puede ser necesario y muy aclaratorio en multitud de situaciones.

Por lo tanto, esta vez y para seguir como forma de trabajo de aquí en adelante, he instalado el IDE de eclipse (neon) con el plugin de maven, para ayudarme en la gestión y construcción de proyectos java.

Con maven puedo gestionar las dependencias del código de forma más eficiente debido a que esa gestión se realiza de forma automática, sin necesidad de que como desarrollador tenga que descargar paquetes e instalarlos como librerías externas. El archivo pom.xml contiene la información necesaria para que en tiempo de "compilación" se acceda a los paquetes que se tienen identificados en el mismo.
Para este ejemplo, el archivo pom.xml es el que incluyo en el anexo I. En caso de aparecer más dependencias en el futuro debido a necesidades de recursos para el desarrollo del trabajo, iré añadiendo entradas de las características necesarias a dicho fichero.

Una vez que se han solucionado las dependencias es conveniente realizar un "maven install" para que sean tenidas en cuenta como dependencias maven. Después se puede realizar un "maven clean" que realiza un build del workspace necesario para la ejecución. Una vez que ya no existen aparentemente errores, se puede realizar un "maven test" para asegurarse de que la ejecución no va a devolver errores debidos al build, no así errores propios del código debido a datos o errores del algoritmo.
Ahora ya tengo todo preparado para ejecutar el código y ver cómo se comunican Kafka y storm según el ejemplo preparado a tal efecto.

Para ello es preciso lanzar de forma manual el servidor de Kafka (incluyendo previamente el lanzamiento de zookeeper) y emitir una serie de mensajes. Estos mensajes serán los que se descompongan en palabras y se mostrará por consola el número de cada una de ellas.

Los pasos a seguir son los que ya se han descrito anteriormente y que vuelvo a incluir. Cada uno de estos comandos se escribirá en un terminal nuevo.
Comandos:

- **Start zookeeper:** bin/zookeeper-server-start.sh config/zookeeper.properties

- **Start Kafka:** bin/kafka-server-start.sh config/server.properties

- **Crear un topic:** bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test-topic

- **lanzar un producer:** e ir escribiendo mensajes que serán los que se lean desde el ejemplo que usa storm y que lo lanzaré desde eclipse: bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test-topic

Una vez que está lanzado el producer de Kafka, escribo los siguientes mensajes que se pueden apreciar en esta imagen:

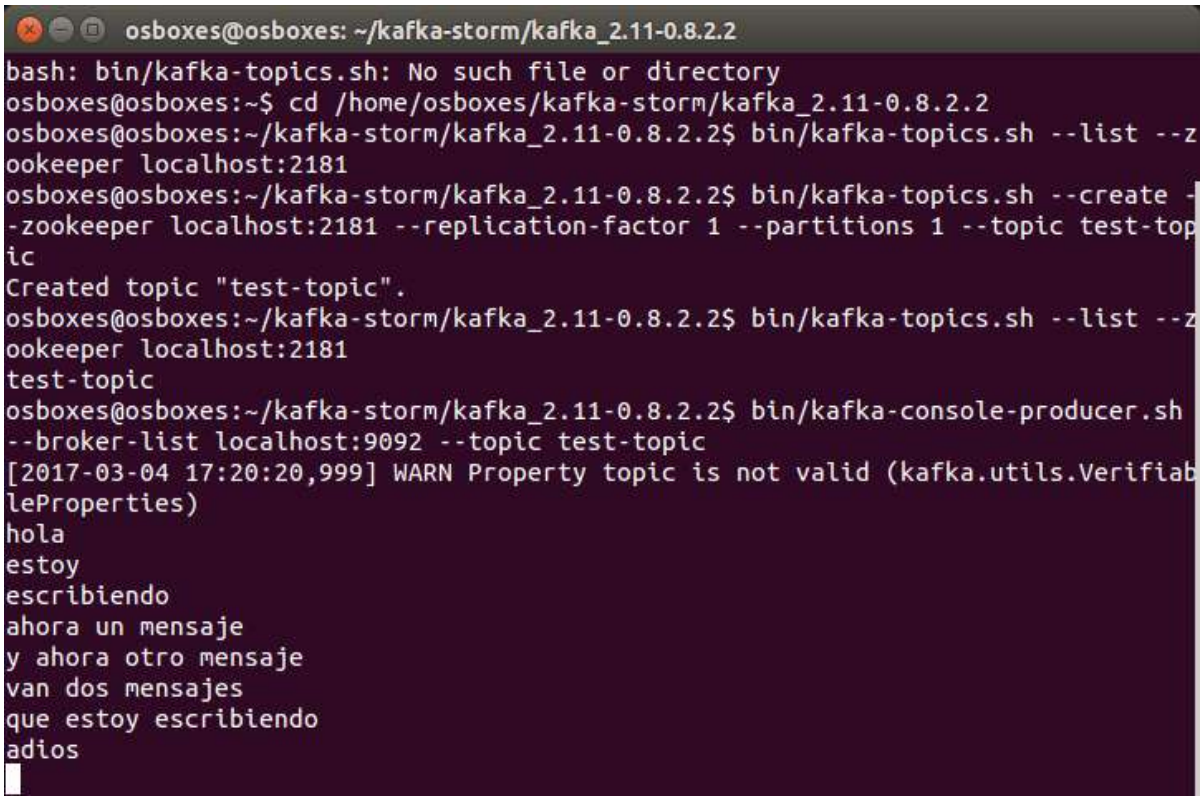

Ilustración 8: Mensajes en un producer

Entonces, desde eclipse, lanzo el ejemplo que he construido para leer los mensajes, y ofrecer como salida un conteo de cuántas veces está una palabra repetida en dichos mensajes.
Para ello, pinchando en cualquier punto del área del editor con ese ejemplo java abierto, pulso el botón derecho y selecciono “Run as” y “java application”.
En ese momento comienza la ejecución del ejemplo, y se van mostrando por la consola de eclipse una serie de mensajes correspondientes a la traza que va dejando dicha ejecución. Puedo apreciar información acerca de los mensajes que se están leyendo, el tratamiento que se está haciendo a esos mensajes y a las palabras que lo componen etc...
Finalmente aparece el resultado esperado y que puede apreciarse en la siguiente imagen:

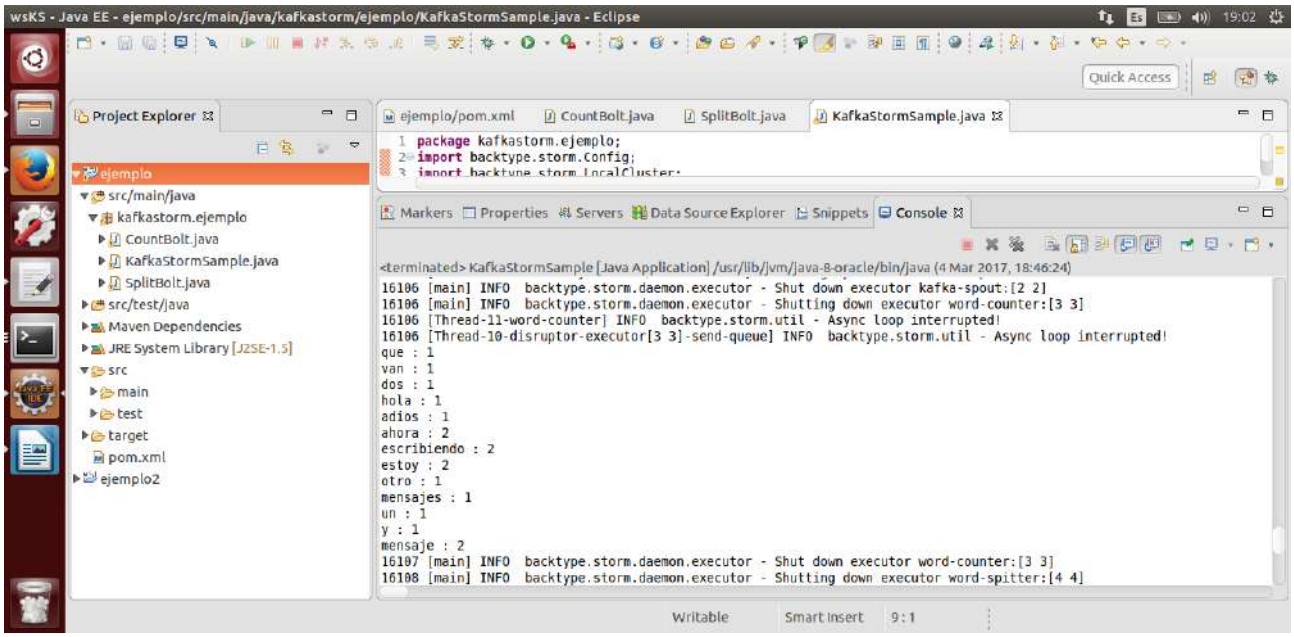

Ilustración 9: Resultado ejemplo conteo palabras

Resultado que es correcto y que puede comprobarse de forma manual contando las palabras indicadas en las frases escritas en el terminal correspondiente al producer de Kafka.

4. Esquema IoT.

He llegado a este punto habiendo probado que es posible la integración de Kafka y Storm, como era previsible. El siguiente paso es, en base a esta plataforma, probada y confirmada como buena en cuanto a emitir datos desde Kafka y recibir y procesar por Storm, imaginar una situación de la vida real en la que este sistema podría usarse y obtener un beneficio del funcionamiento del mismo.

4.1 Premisas y objetivos

A la hora de imaginar y establecer un contexto en el que implantar y aplicar la plataforma creada, comienzo por pensar en situaciones generales, dentro de una ciudad. Como por ejemplo una ciudad donde existen una serie de sensores que miden la velocidad del viento en ese punto:

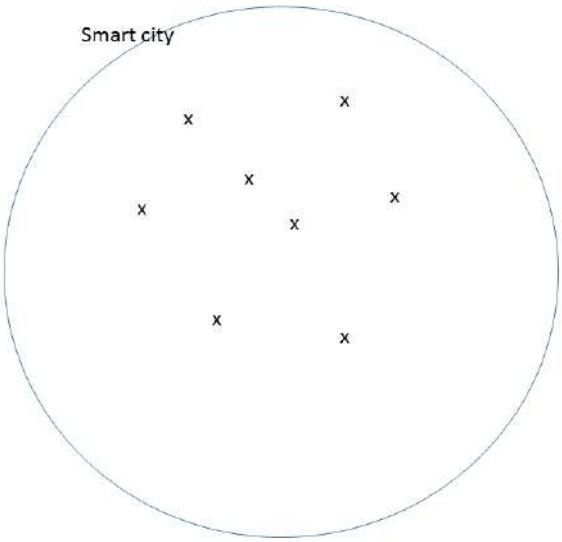

Ilustración 10: Sensores un una SmartCity

Los puntos x identifican los diferentes sensores que se encuentran diseminados por la ciudad, suponiendo que esta ciudad es una ciudad inteligente o Smart City.
Podemos pensar en implementar un modelo que recoja los valores que miden estos sensores, tomando un conjunto de medidas válidas y en base a esas medidas emitir una valoración, que puede ser una media de las mismas o algún otro tipo de resultado mediante un tratamiento de los datos, que pueda aportar alguna información útil.

Las medidas se podrán tomar como válidas por ejemplo si están en un intervalo de valores previamente identificado (If (x<=medida<=y) entonces la medida se considera válida)

Podría tomarse como base este modelo y una vez diseñado, evolucionarlo con una serie de información a obtener del mismo que puede convertirse en un sistema útil para la ciudadanía, mediante un análisis a realizar a las medidas que lo nutren. A modo de ejemplo, esa información podría ser:

- Número de sensores necesarios para obtener medidas fiables según unos parámetros pre-establecidos
- Lugar óptimo de colocación de los sensores para obtención de medidas diferenciadas y que aporten valor.
- Cuál puede ser la mejor conexión/conectividad de los sensores (wifi, 4G, internet,..), o evaluarla según lo implementado.
- Fuente de energía (red, baterías, energías renovables)
- …..

Este conjunto de sensores en una smartcity o ciudad inteligente, consistiría en una red estable, que enviaría la información a un centro de datos (cloud) donde se ejecutarían los algoritmos necesarios.

Estos nodos, podrían estar situados cada uno en un “barrio” de la ciudad, agrupando así a los sensores por su localización, y estableciendo una similitud entre la tipología de la red de sensores y la estructura geográfica de la ciudad.

Completamos el esquema de la red con un o unos nodos intermedios que pueden realizar un proceso simple de filtrado o preparación de la información para que sea procesada en cloud, donde se muestra las conclusiones (información final).
Este centro de proceso lo vamos a llamar el CORE.

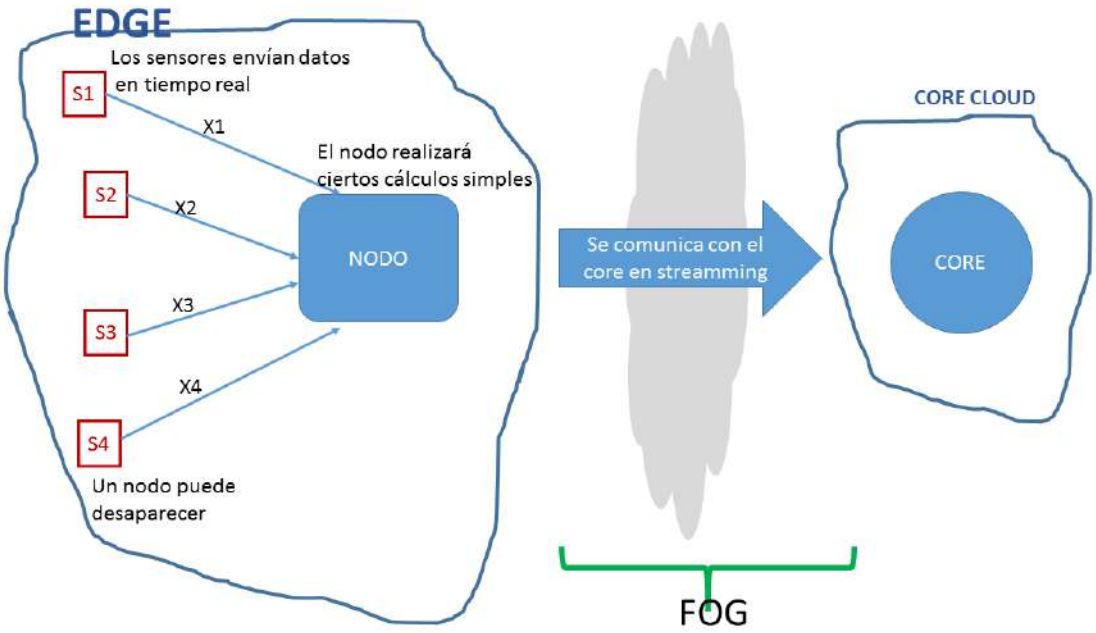

Ilustración 11: Esquema de red

Este modelo podría aplicarse para múltiples propósitos, por ejemplo, si los sensores midieran movimientos sísmicos, podría establecerse que los datos que se recogen son los necesarios, en tiempo real, para emitir avisos ante un peligro de sufrir terremotos en la ciudad.

Si en la figura planteada anteriormente, hiciéramos un zoom en la parte en la que se comunican el nodo y el core, podríamos establecer este esquema de diseño de esta parte del modelo:

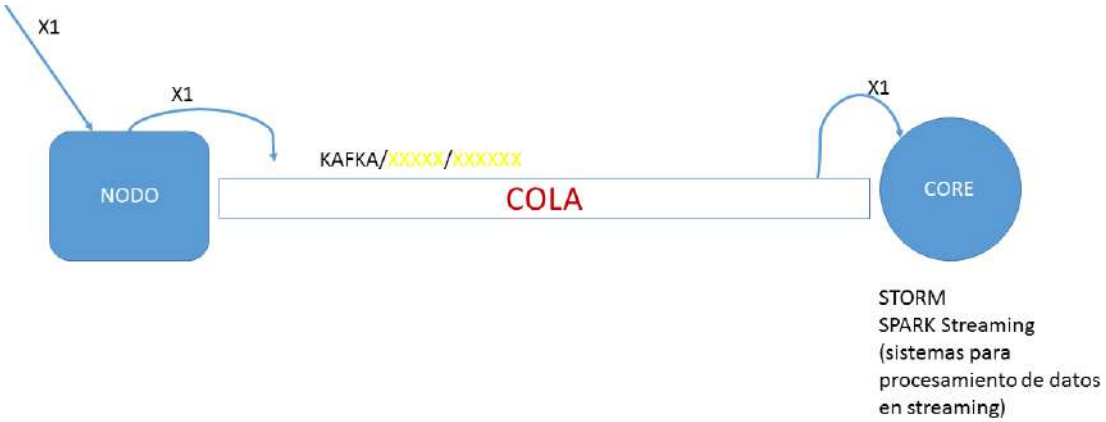

Ilustración 12: Zoom comunicación con el core

Este zoom al modelo que puede verse en la imagen superior, delimita el alcance de Esta implementación de esta parte amplificada del modelo, será el comienzo del desarrollo del trabajo a realizar dentro del master.

A grandes rasgos, consiste en recoger la información de los sensores hacia el nodo, de allí redirigir esa información hacia el core usando Kafka y una vez recogidos esos datos por el core, y mediante un sistema de procesamiento de datos en streaming (storm, spark) realizar los algoritmos que se consideren necesarios (parte de analytics) para obtener la información final deseada.

En la realización de este trabajo, no dispongo de sensores que pueda usar para recoger los datos que emitan, por lo que voy a construir una serie de emuladores de productores de datos, en la forma de conjunto de procesos donde cada cierto tiempo (1 segundo por ejemplo) se emitirá un dato aleatorio de velocidad del viento, dato que se irá añadiendo a la cola para ser procesados en Kafka.

Esos datos, una vez introducidos en el sistema, van superando una serie de pasos en la cola que hacen que se transfieran a kafka. Todo este proceso lo implemento en java, mediante el IDE de eclipse.
La información que llega al core, puede almacenarse para usarse más adelante en conjunción con otros datos (ejecutar otro tipo de analytics, HDFS,…), devolver los datos procesados al edge, o trasladarlos a los usuarios.
Un ejemplo de procesamiento podría ser detectar de cada 1000 datos, si hay un x% que superan un umbral y que en caso afirmativo, en algún lugar suene una alarma. También se puede medir la latencia de los datos, es decir, el tiempo que transcurre desde que ese dato es emitido, hasta que es almacenado y procesado.

4.2 Desarrollo.

Los elementos software que he desarrollado son los que se indican en el esquema siguiente:

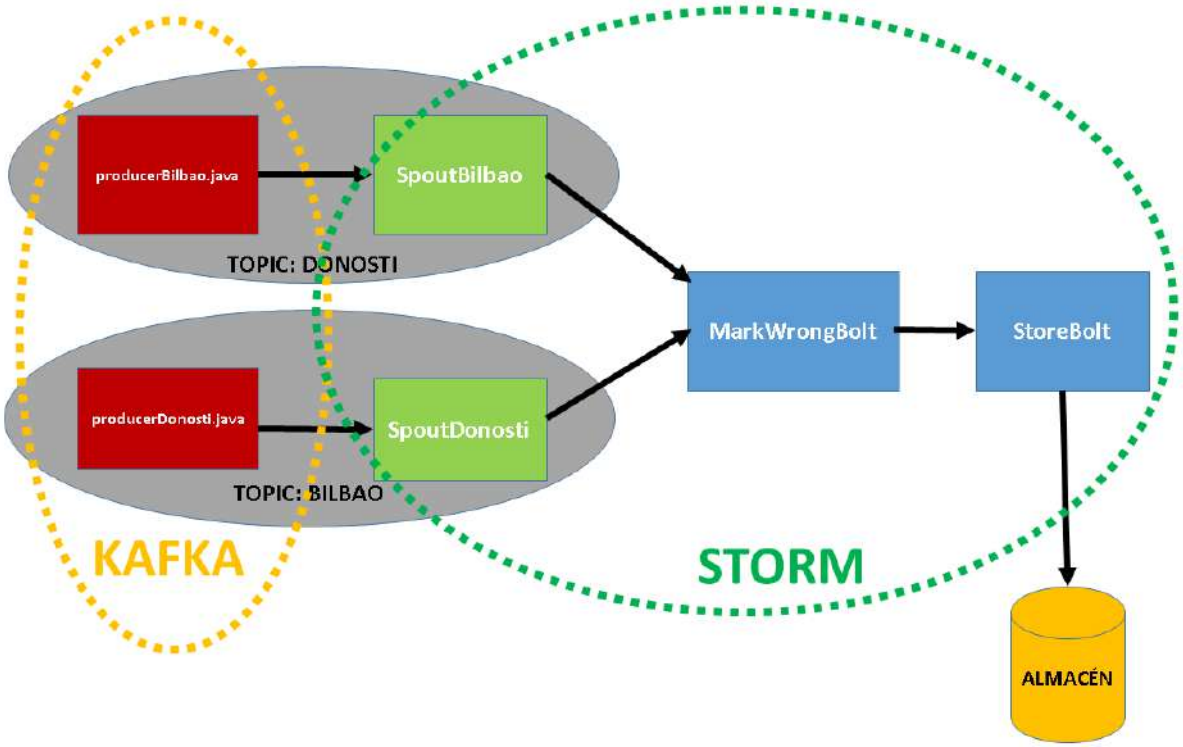

Ilustración 13: Arquitectura problema Iot y BigData

a) **ProducerBilbao.java:** Código java que lanza datos random al sistema, simulando ser valores de medición de viento en la ciudad de Bilbao. Para ello, lanza esos datos al topic Bilbao creado previamente dentro del sistema Kafka. Esta implementación se realiza creando un nuevo producer de tipo KafkaProducer que es un API que provee la instalación Kafka por defecto. Una vez creado el objeto, se le asigna una serie de propiedades, como por ejemplo:

```
//Assign localhost id

    props.put("bootstrap.servers", "localhost:9092");

    //Specify buffer size in config

    props.put("batch.size", 16384);

    //The buffer.memory controls the total amount of memory
available to the producer for buffering.

    props.put("buffer.memory", 33554432);

  props.put("key.serializer",


"org.apache.kafka.common.serialization.StringSerializer");
```

Etcétera…..

Una vez creado el objeto producer, construyo los datos a emitir. Estos datos los genero de forma aleatoria, y le doy formato de objeto json1, con la siguiente estructura:

```
JSONObject windmeasures = new JSONObject();

        windmeasures.put("timestamp", new Timestamp
(System.currentTimeMillis()).getTime());

        windmeasures.put("location", topicName);

        windmeasures.put("velocity", Math.random() * 80);
```

El dato que se transmitirá desde el producer, lleva asociados los valores de timestamp, o momento en el que se crea, location o topic o ciudad de la que procede el dato, y el propio dato de velocidad del viento, siendo este último dato el único creado de forma aleatoria.

Una vez dispongo del dato a enviar, se envía mediante el método send del producer, y espero un cierto tiempo (5 segundos) hasta enviar el siguiente dato:

```
producer.send(new ProducerRecord<String,
String>(topicName,Integer.toString(i),windmeasures.toString()))
;

       Thread.sleep(5000);
```

Mediante un bucle, produzco 100 datos a emitir, para simular un sistema que está continuamente produciendo, recibiendo y procesando. De esta forma puedo analizar los datos y el funcionamiento del sistema de forma controlada.

b) **ProducerDonosti.java:** De igual forma que con el producer anterior, se generan datos que simulan ser mediciones de la velocidad del viento en la ciudad de Donosti. El código es similar, cambiando el nombre del topic de Bilbao a Donosti.

c) **SpoutBilbao y SpoutDonosti**: Estos elemento son los spouts por defecto de Storm. No he realizado ninguna modificación en ellos. Se crean en el código java que genera toda la topología del sistema Storm. Estos elementos únicamente realizan la función de introducir los datos que se reciben de Kafka, desde los producers, y los introducen en el sistema Storm. Se les asigna unos valores de configuración en su creación (instancia) entre los que se encuentra el nombre del topic del que recogen la información.

[1] JavaScript Object Notation. Sintaxis para almacenar e intercambiar datos.

d) **MarkWrongbolt**: Este elemento es un Bolt de la topología de Storm, que son los elementos que realizan el procesamiento dentro de una topología.
En este caso, y en el siguiente, realizo una extensión de la interface de Storm IRichBolt.
Dentro del método execute es donde reside toda la inteligencia del proceso, por lo que este método tiene que ser sobrescrito mediante @override.
El proceso es el siguiente:
Si el campo velocity es mayor que 50, se marca como valor erróneo, y se le asignan los valores correspondientes a wrongmeasures (número de valores erróneos hasta el momento) y evaluated (número de datos evaluados). Esto por cada ciudad o topic.
Se completa el dato en formato json con estos valores y se vuelve a emitir para que el siguiente elemento del sistema lo recoja y realiza el siguiente procesamiento.

e) **StoreBolt**: Este elemento se encarga de recoger los datos procesados del Bolt que le precede y almacenarlos tras realizar otro pequeño procesamiento.
El método execute se sobrescribe con el siguiente proceso:
Con cada dato que se recibe, se analiza si es un dato correcto (no incluye la propiedad “wrong”) y si es así se calcula la latencia, restando del momento actual, el valor almacenado en el campo “timestamp” del objeto recibido. Este valor se almacena también en el mismo objeto creando para ello una nueva propiedad y asignándosela al dato. Además se escribe en un fichero previamente creado, de nombre wind+topic.txt, el dato con todos sus campos.
En caso de que el dato fuera erróneo, la información que se vuelca en el fichero es el número de datos erróneos hasta el momento respecto del total de evaluados.

f) **WindMeasures**: Este elemento es el elemento principal, que lanza todo el sistema y lo mantiene funcionando. Aunque para la realización del ejemplo mantengo funcionado esta topología durante 15 minutos (900.000 ms), un sistema real estaría funcionando de forma continua.
El proceso que he incluido en este elemento es el siguiente:
Mediante consola se piden al usuario los nombres de dos ciudades o topics. Esto en una situación real quizá no sea muy útil pero en mi experimento me permite darle el nombre que quiero en cada momento.
A continuación, creo dos Spout con la misma configuración pero con distinto topic name.

Con estos dos elementos creados y las clases java con los Bolt creados anteriormente de forma separada, montamos la topología necesaria para que mi sistema funcione. Esto se realiza definiendo la estructura y el orden en el que entrarán a funcionar los distintos elementos.

```
TopologyBuilder builder = new TopologyBuilder();
```

A la topología creada, se le asignan los dos spouts, uno por cada localización:

```
      builder.setSpout("spout-location1", new
KafkaSpout(Spout1Config));
      builder.setSpout("spout-location2", new
KafkaSpout(Spout2Config));
```

Ahora se asignan los Bolts, en el orden correcto, para que un dato pase primero por el proceso de marcar los valores erróneos y después por el almacenamiento de los mismos, nunca al revés. Se indica además que al Bolt MarkWrongBolt le entren los datos de los dos Spouts definidos, y en cambio al Bolt StoreBolt le lleguen como entrada los datos que emita el Bolt MarkWrongBolt (o "wrong-measures" ya que en la asignación de orden en la topología, se asignan otros nombres a estos elementos)

```
      builder.setBolt("wrong-measures", new
MarkWrongBolt()).shuffleGrouping("spout-
location1").shuffleGrouping("spout-location2");
      builder.setBolt("store", new
StoreBolt()).shuffleGrouping("wrong-measures");
```

Una vez creada la estructura de la topología, hay que lanzar. Esto se realiza desplegando la topología sobre un cluster, que en este experimento será un cluster local por razones de limitación de recursos.

```
      LocalCluster cluster = new LocalCluster();
      cluster.submitTopology("WindMeasures", config,
builder.createTopology());
```

Y en este punto ya está listo el sistema para funcionar. No obstante, para que no se gasten todos los recursos de mi instalación, dejo 15 minutos el sistema corriendo y en ese momento lo paro. Esto se realiza mediante el método shutdown del cluster:

```
      Thread.sleep(900000);
      cluster.shutdown();
```

El código de estos elementos se encuentra en el ANEXO II.

4.3 Puesta en marcha.

Una vez que tengo construido todo el código necesario para hacer funcionar el sistema ideado, sobre la plataforma creada previamente, sigo los siguientes pasos para lanzarlo y comprobar cómo viajan los datos desde los producers hasta el almacenaje final.
Los pasos a seguir son:

- **Start zookeeper:** bin/zookeeper-server-start.sh

config/zookeeper.properties

- **Start Kafka:** bin/kafka-server-start.sh config/server.properties

- **Crear el topic Bilbao:** bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Bilbao

- **Crear el topic Donosti:** bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic Donosti

- **Lanzar desde eclipse ProducerBilbao.java:** botón derecho, run as, java application.

- **Lanzar desde eclipse ProducerDonosti.java:** botón derecho, run as, java application.
- **Lanzar desde eclipse WindMeasures.java:** botón derecho, run as, java application.

  Me aparece en este momento por consola la petición de introducir el nombre de un topic. Introduzco Bilbao y me aparece la petición de introducir otro nombre de topic, momento en el que indico Donosti:

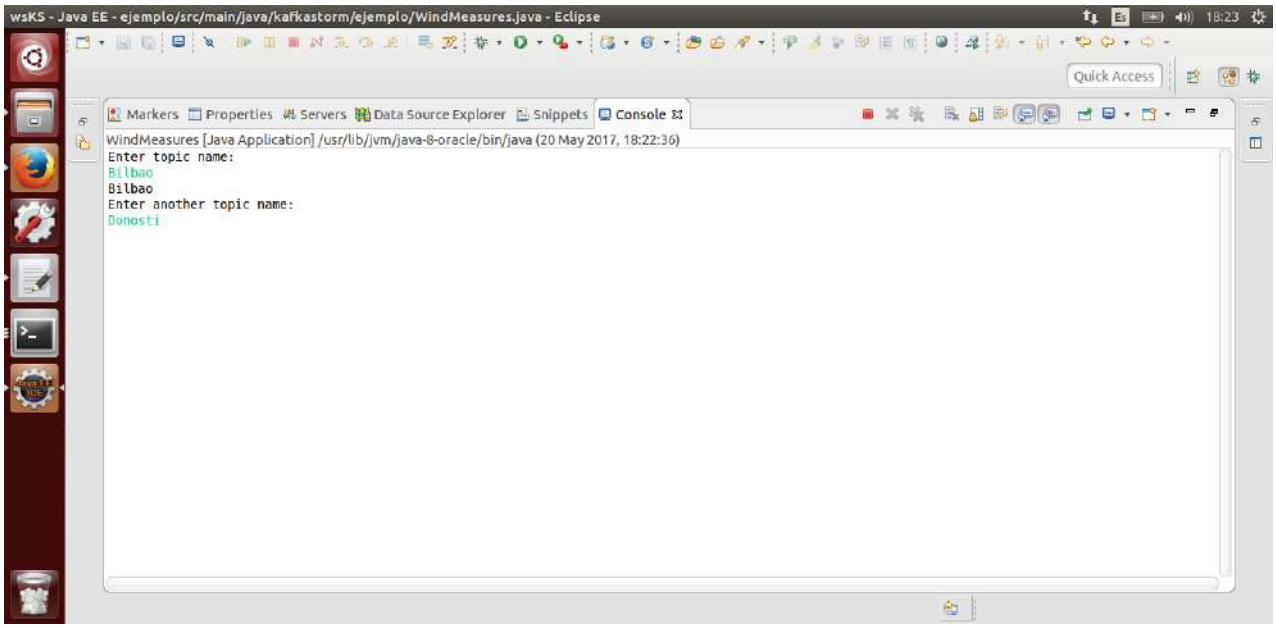

Ilustración 14: Despliegue cluster Storm. Consola.

A partir de este momento, por esta misma consola de eclipse se van mostrando las distintas trazas que desde Storm se emiten, relativas a las acciones que se están realizando en cada momento.

Después de transcurridos los 15 minutos establecidos desde que se lanzó la topología, el proceso termina y las últimas trazas que se muestran por consola son las siguientes:

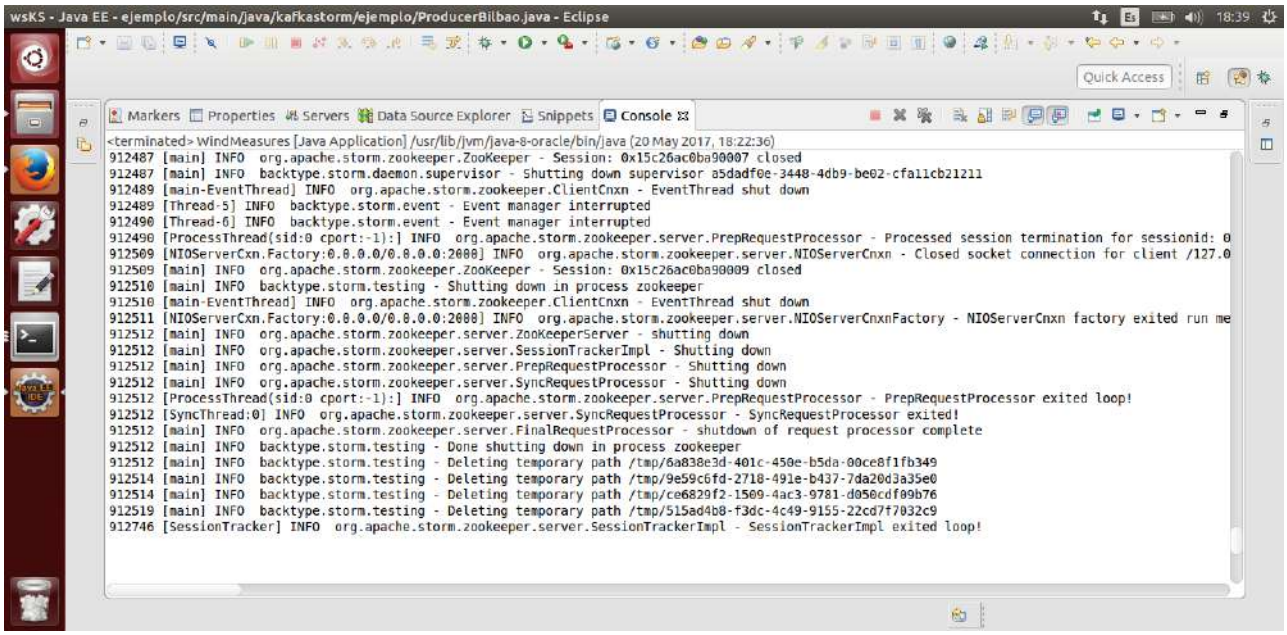

Ilustración 15: Traza fin ejecución sistema completo

Ahora sólo falta comprobar que los valores se han almacenado correctamente, tal y como se ha indicado en el elemento StoreBolt.

Estos ficheros se encuentran en el ANEXO III, pero si analizamos los primeros valores en el fichero correspondiente a la ciudad Bilbao, podemos ver:

```
{"latency":121233,"location":"Bilbao","velocity":23.02712517127783,"timestamp":1495297320117}
```

Este primer dato almacenado tiene un valor de velocidad del viento de 23.03 m/s
Y el timestamp del momento en el que se originó que es `1495297320117`.
La latencia es de 121233 ms que es el tiempo transcurrido entre que se originó el valor y se almacenó.

```
Bilbao: Llevamos 1 medidas mayores de 50, de entre: 2 evaluadas.
```

La línea siguiente no se corresponde con un valor de medición de viento, eso quiere decir que el segundo valor emitido por el producer Bilbao, era un valor erróneo según el baremo que se tiene en cuenta en el sistema, esto es, que era mayor a 50 m/s.
Se indica entonces que ha habido una medida mayor de 50, de entre 2 evaluadas en total, para ese topic o ciudad.

……

Una vez terminado el objetivo principal, y tras conversaciones con mi director en las que nos planteábamos avanzar en cuanto al almacenamiento de los resultados en algún sistema distribuido que pudiera dotar de mayor flexibilidad

al sistema completo, opté por comprobar si este planteamiento era fácilmente integrable en una solución de almacenaje HDFS.
Para ello, a modo de comprobación de dicha ampliación en el flujo de datos, realicé una instalación en mi máquina virtual en local, de un HDFS single node. No es mi intención explorar todas las posibilidades que este sistema me puede aportar, sino demostrar que mi infraestructura es fácilmente integrable y compatible con otros sistemas de almacenaje, como por ejemplo HDFS. Un sistema real, a gran escala, requeriría de una instalación distribuida, en distintas máquinas y realizando un acceso remoto al servicio de almacenamiento instalado.

Por lo tanto, he realizado unas modificaciones a la topología de Storm lanzada anteriormente para introducir un elemento, Bolt en este caso, que será el encargado de conectar con el sistema HDFS y almacenar allí los resultados generados por mi sistema.
La arquitectura de elementos por lo tanto queda modificada de la siguiente forma:

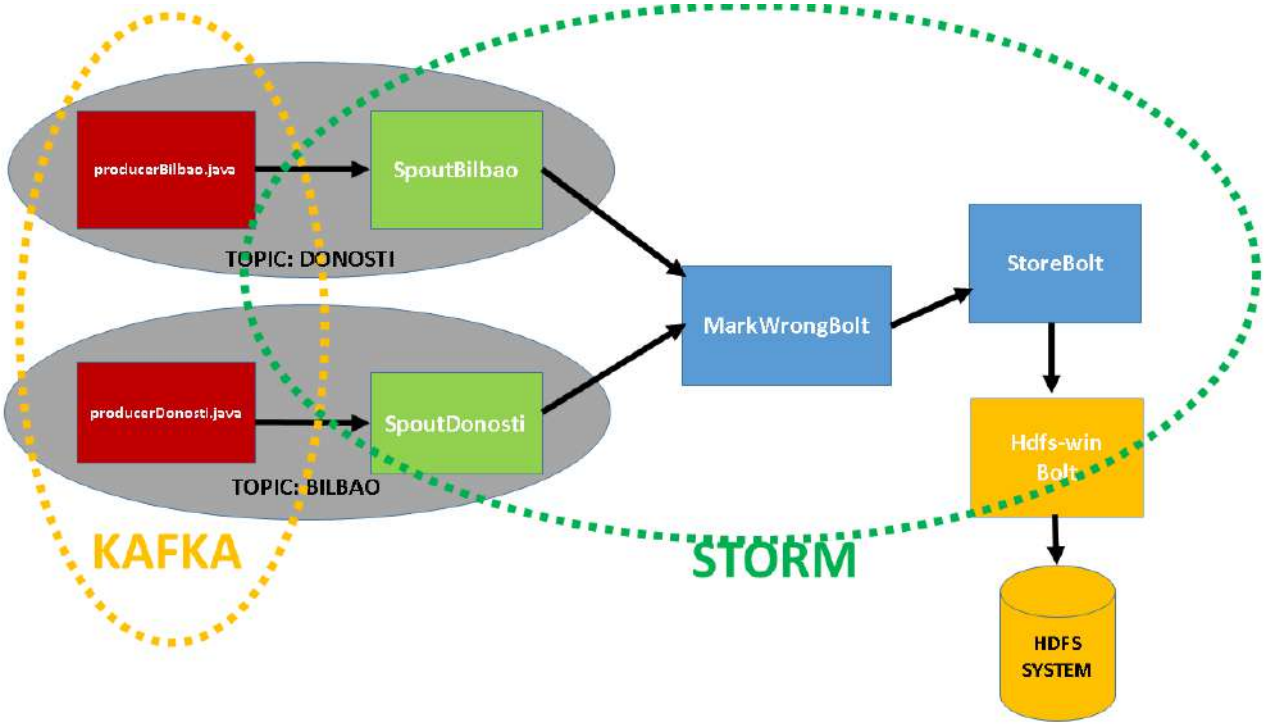

Ilustración 16: Arquitectura de elementos plataforma con hdfs

Y el código añadido para incluir este elemento en el sistema se encuentra en el elemento java WindMeasures.java de esta forma:

```
      RecordFormat format = new
DelimitedRecordFormat().withFieldDelimiter(",");

      SyncPolicy syncPolicy = new CountSyncPolicy(1000);

      FileRotationPolicy rotationPolicy = new
FileSizeRotationPolicy(127.0f, FileSizeRotationPolicy.Units.MB);
```

```
      DefaultFileNameFormat fileNameFormat = new
DefaultFileNameFormat();

      //The files are written in this HDFS folder

      fileNameFormat.withPath("/user/osboxes");

      //Files start with the following filename prefix

      fileNameFormat.withPrefix("wind");

      //Files end with the following suffix

      fileNameFormat.withExtension(".txt");


      //HDFS bolt

      HdfsBolt bolt =

          new HdfsBolt().withFsUrl("hdfs://localhost:9000")

              .withFileNameFormat(fileNameFormat)

              .withRecordFormat(format)

              .withRotationPolicy(rotationPolicy)

              .withSyncPolicy(syncPolicy);
```

Y una vez asignadas las propiedades necesarias al Bolt creado, para que se conecte al sistema HDFS, y gestione la información en ficheros de las características descritas, se añade ese elemento a la topología, en el lugar que le corresponde, que es en este caso detrás del bolt StoreBolt, del que va a recibir información.
En ese Bolt he tenido que realizar una modificación consiste en incluir la instrucción emit para trasladar la información que se desea almacenar. Como este Bolt era el último en la cadena, no necesitaba emitir cadenas de datos, y por ello esta instrucción no estaba incluida.

Las instrucciones para esta nueva topología son:

```
TopologyBuilder builder = new TopologyBuilder();

      builder.setSpout("spout-location1", new
KafkaSpout(Spout1Config));

      builder.setSpout("spout-location2", new
KafkaSpout(Spout2Config));

      builder.setBolt("wrong-measures", new
MarkWrongBolt()).shuffleGrouping("spout-
location1").shuffleGrouping("spout-location2");
```

```
      builder.setBolt("store", new
StoreBolt()).shuffleGrouping("wrong-measures");

      builder.setBolt("hdfs-wind", bolt).fieldsGrouping("store", new
Fields("loc"));
```

Para arrancar el Sistema hdfs, abro un terminal y ejecuto la siguiente instrucción:

sbin/start-dfs.sh

y para comprobar que está arrancado, desde un navegador me conecto a http://localhost:50070/ donde se puede ver lo siguiente:

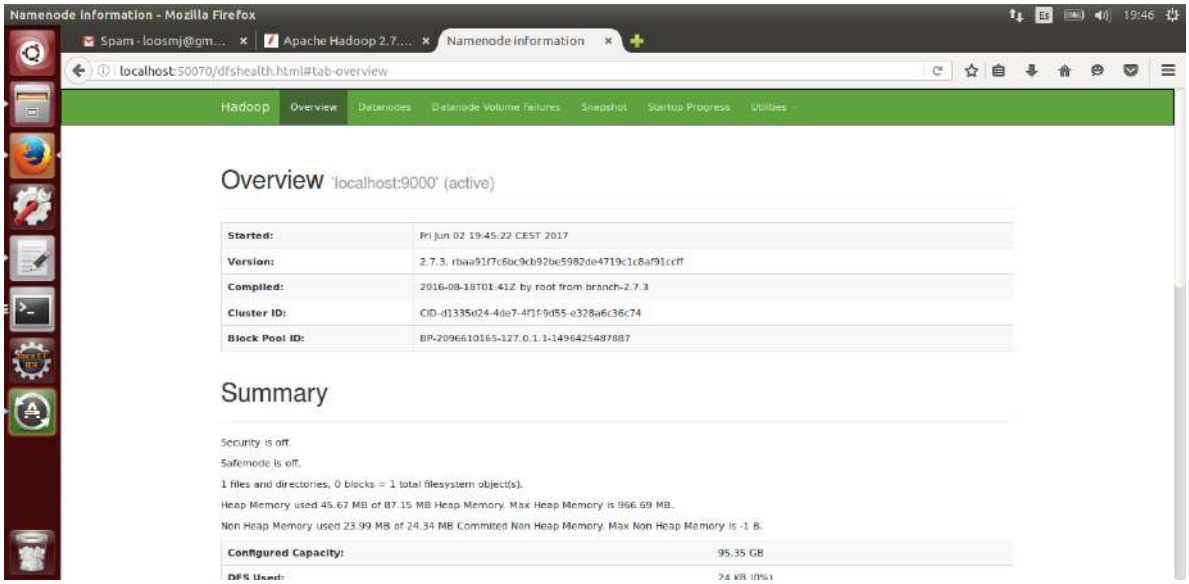

Ilustración 17: información sobre sistema hdfs

Ahora creo la estructura de directorios que considero necesaria para mi ejemplo:

dfs -mkdir /user
dfs -mkdir /user/osboxes

Y he creado la estructura que puede verse en el navegador:

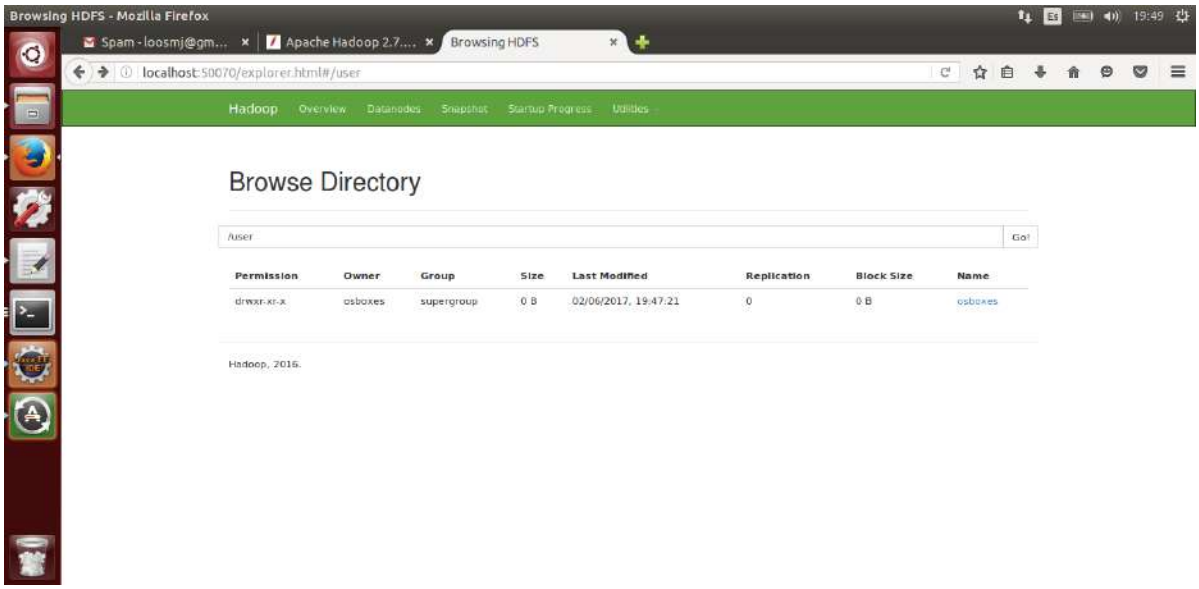

Ilustración 18: Sistema de directorios en instalación hdfs

Después de la ejecución del sistema integrado al completo, en el directorio creado aparece el fichero con los datos almacenados acerca de las mediciones:

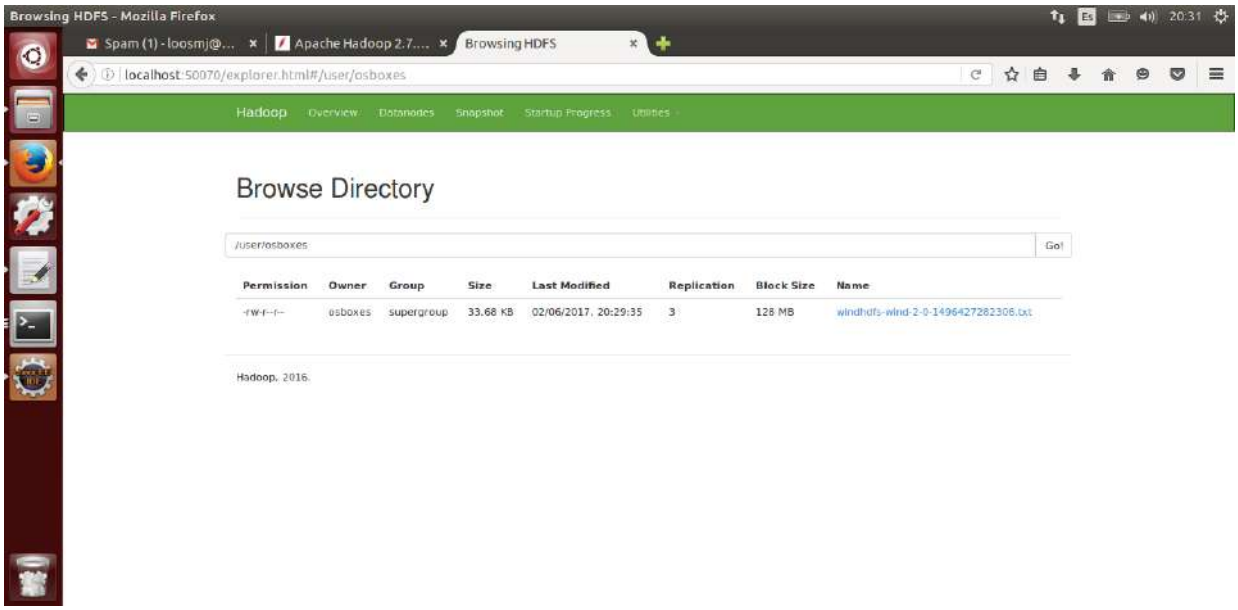

Ilustración 19: Fichero creado y almacenado en hdfs como output del sistema completo

Concretamente la información acerca de ese fichero es la siguiente:

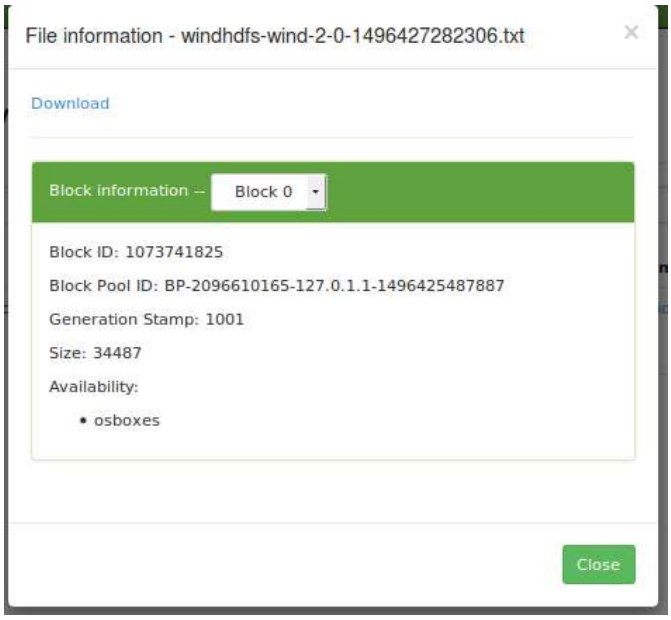

Ilustración 20: Información fichero almacenado en hdfs

En el anexo II se puede ver el contenido completo del fichero.

5. Siguientes pasos.

Una vez que la plataforma está completa y encuentro cumplidos los objetivos, se ve claramente cómo establecer un proyecto de mayor alcance que el desarrollado, por lo que en los puntos que incluyo a continuación describo los pasos a dar para escalar esta plataforma.

5.1 Situación actual

Según la exposición del sistema que he realizado anteriormente, puedo darme cuenta de que este experimento es una prueba de concepto, útil para mi aprendizaje y para confirmar que es posible implementar este tipo de plataformas desde cero, sin conocimientos profundos previos.

Aunque la solución aportada es simple, me ha permitido atisbar las enormes posibilidades de este tipo de soluciones en los casos en los que los recursos no son un problema y los conocimientos son más amplios, además de realizar el trabajo por parte de un equipo.

Tanto la instalación de Kafka como la de Storm, han sido instalaciones que simulan una estructura distribuida, pero que en realidad están desplegadas dentro de la misma máquina. Sólo he arrancado un servidor de Kafka, los topics solamente constan de 1 nivel de réplica, los datos son simulados, el cluster de Storm es un cluster local, etc….

Si todos esos elementos se utilizan de forma experta, distribuida realmente, con objetivo de sacar el mayor partido, el potencial podría ser enorme.

5.2 Situación a la que escalar

El objetivo podría ser un sistema, en el que se provea a los usuarios de determinados clubes de deportes de vela, las mejores playas en cada momento donde disfrutar al máximo del deporte que les apasiona.
Estos usuarios podrían elegir unas cuantas playas, que serían las que están dentro de un radio al que se puedan desplazar durante ese día, y elegir la que tenga mayores velocidades del viento en las últimas horas.
Para ello, se habrán instalado una serie de sensores en cada playa, que emitirán a sistemas Kafka (varios servidores arrancados), donde se asociarán a topics (los datos que se manejarían serían muy numerosos y tendrían que tener valores de réplica grandes), a los que se subscribirán los distintos elementos de Storm. Este Storm estará desplegado en un cluster en “Producción”, que es el nombre que se le da al cluster cuando está compuesto de muchos procesos, que corren en diferentes máquinas y que continúan corriendo hasta que se paran de forma deliberada.

Otra aplicación podría ser para aeropuertos o pistas de despegue no profesionales, precisamente porque estarían interesados en conocer los valores de la velocidad del viento para evitar aterrizar en ellos.

Si tuviera que realizar una segunda versión de este trabajo, añadiendo funcionalidad a esta prueba, creo que comenzaría por conseguir que alguna instalación de sensores que estuviera disponible, me dejara acceder a ellos y enganchar de esta forma la obtención de datos reales con mi sistema. Eliminaría los producers hechos en java y los sustituiría por algún tipo de proceso que seguro tendría que realizar a los datos recibidos de los sensores. Investigar cómo introducir esos datos en el sistema y una vez hecho, comprobar si el resto de los procesos sigue siendo válido.
El siguiente paso sería desplegar Storm en dos máquinas distintas. Supongo que este paso requeriría de mucho esfuerzo de estudio y de pruebas de funcionamiento hasta dar con la solución.

No obstante existen soluciones que aúnan estas plataformas para dar un beneficio superior, y que actualmente están de actualidad. Por ejemplo Hortonworks, empresa de santa Clara (California) ofrece soluciones para la creación, distribución y soporte a las empresas de plataformas de datos modernas y sus aplicaciones. Trabaja con comunidades de software libre como Apache Hadoop, NiFi, y Spark, y aseguran que sus plataformas conectadas permiten sostener aplicaciones de datos modernas que distribuyen inteligencia útil obtenida de los datos, tanto en tiempo real como datos almacenados.

6. Conclusiones

La principal conclusión que he obtenido ha sido que es posible desplegar un problema de IoT, integrando plataformas asociadas al mundo del BigData.
De hecho son el complemento perfecto, y se dan sentido mutuamente. La IoT proporciona los datos, permite asociar metadatos a esos datos como por ejemplo de dónde vienen, tiempos de latencia etc… y estos datos, se refinan, se combinan, se procesan y se obtienen ya no datos sino información útil y valiosa para la toma de decisiones, que es en última instancia el fin último del análisis de datos, sean pocos o sean grandes como en este ambiente de BigData.

He aprendido también que los tutoriales y las páginas de “gettingStarted” de muchas tecnologías existentes actualmente, no son lo que parecen. Aportan información muy valiosa, pero no completa y el trabajo de búsqueda de la resolución de los problemas es arduo, aunque gratificante al mismo tiempo. Es esperable debido a que son tecnologías que aunque puedan llevar un tiempo en el mundo, lo han hecho dentro de los ambientes de investigación y por ello no se han probado en muchos aspectos o casos.

He descubierto que maven es de una utllidad muy grande a la hora de gestionar las librerías que se necesitan en un proyecto. Permite no tener que buscarlas, bajarlas, añadirlas, probarlas de forma manual, sino que mediante una indicación en un fichero de configuración (pom.xml) maven lo hace por ti. Me ha ahorrado mucho tiempo y muchos problemas.

El método seguido para la realización del trabajo, propuesto por mi director, ha sido el adecuado, ya que me ha permitido ir superando etapas y sobre esas añadir funcionalidades que incrementaran la solución en el camino que había decidido. Este método ha consistido en comenzar con problemas pequeños, que pudiera conocer cómo atacar, y que una vez solucionados, pudiera ir cubriendo con otra capa para avanzar.

7. Bibliografía

Para la realización de esta memoria y del trabajo, se han consultado las siguientes fuentes:

[1] pacomaroto.wordpress.com/about/introduccion-a-la-internet-de-las-cosas/
[2] iot-analytics.com/10-internet-of-things-applications/
[3] https://www.sas.com/en_us/insights/big-data/what-is-big-data.html
[4] www.huffingtonpost.com/philip-kushmaro/the-iot-and-big-data-maki_b_12116608.html
[5] www.kdnuggets.com/2016/09/big-data-iot-match-made-heaven.html
[6] en.wikipedia.org/wiki/Apache_Kafka
[7] kafka.apache.org/intro
[8] www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm
[9] www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm
[10] kafka.apache.org/intro
[11] storm.apache.org/
[12] storm.apache.org/releases/current/Tutorial.html
[13] www.tutorialspoint.com/apache_storm/apache_storm_core_concepts.htm
[14] en.wikipedia.org/wiki/Apache_Hadoop
[15] hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction
[16] www.malinga.me/reading-and-understanding-the-storm-ui-storm-ui-explained/

ANEXO I: Maven. Fichero pom.xml para ejemplo de conteo de palabras

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>kafkastorm</groupId>

  <artifactId>ejemplo</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <packaging>jar</packaging>

  <name>ejemplo</name>

  <url>http://maven.apache.org</url>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  </properties>

  <dependencies>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>3.8.1</version>

      <scope>test</scope>

    </dependency>

    <dependency>

        <groupId>org.apache.storm</groupId>

        <artifactId>storm-core</artifactId>

        <version>0.9.7</version>

    </dependency>

   <dependency>

        <groupId>org.apache.curator</groupId>

        <artifactId>curator-client</artifactId>
```

```
            <version>2.9.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-framework</artifactId>
            <version>2.9.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.storm</groupId>
            <artifactId>storm-kafka</artifactId>
            <version>0.9.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.8.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.11</artifactId>
            <version>0.8.2.2</version>
        </dependency>
    </dependencies>
</project>
```

ANEXO II: Elementos software esquema IoT

ProducerBilbao.java

```
package kafkastorm.ejemplo;

import java.sql.Timestamp;
import java.util.Properties;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

import org.apache.kafka.clients.producer.Producer;

import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerRecord;
import org.json.simple.JSONObject;

//Create java class named SimpleProducer
public class ProducerBilbao {

 public static void main(String[] args) throws Exception{
 String topicName;
 org.apache.log4j.BasicConfigurator.configure();
    // Check arguments length value
    if(args.length == 0){
        topicName= "Bilbao";
       System.out.println(topicName);
    }
    else {
        //Assign topicName to string variable
       topicName = args[0].toString();

    }


    // create instance for properties to access producer configs
    Properties props = new Properties();

    //Assign localhost id
    props.put("bootstrap.servers", "localhost:9092");

    //Set acknowledgements for producer requests.
    props.put("acks", "all");

    //If the request fails, the producer can automatically retry,
    props.put("retries", 0);

    //Specify buffer size in config
    props.put("batch.size", 16384);

    //Reduce the no of requests less than 0
    props.put("linger.ms", 1);

    //The buffer.memory controls the total amount of memory available
to the producer for buffering.
    props.put("buffer.memory", 33554432);

    props.put("key.serializer",
       "org.apache.kafka.common.serialization.StringSerializer");
```

```
    props.put("value.serializer",
       "org.apache.kafka.common.serialization.StringSerializer");

    Producer<String, String> producer = new KafkaProducer
       <String, String>(props);

    Timestamp ts = new Timestamp (System.currentTimeMillis());
    // en lugar de estar produciendo indefinidamente, voy a hacer que
produzca 100 valores
    int times=0;
    for(int i = 0; i < 100; i++)
   {
        JSONObject windmeasures = new JSONObject();
        windmeasures.put("timestamp", new Timestamp
(System.currentTimeMillis()).getTime());
        windmeasures.put("location", topicName);
        windmeasures.put("velocity", Math.random() * 80);


       producer.send(new ProducerRecord<String,
String>(topicName,Integer.toString(i),windmeasures.toString()));
       //System.out.println("dato nÂ°: "+i);
       //System.out.println(windmeasures);
       Thread.sleep(5000);
       times=i;
    }
    System.out.println("Message sent successfully "+times+" times");
             producer.close();
 }
}
```

ProducerDonosti.java

```
package kafkastorm.ejemplo;

import java.sql.Timestamp;
import java.util.Properties;
import java.util.Scanner;

//import simple producer packages
import org.apache.kafka.clients.producer.Producer;

//import KafkaProducer packages
import org.apache.kafka.clients.producer.KafkaProducer;

//import ProducerRecord packages
import org.apache.kafka.clients.producer.ProducerRecord;
import org.json.simple.JSONObject;

//Create java class named SimpleProducer
public class ProducerDonosti {

 public static void main(String[] args) throws Exception{
 String topicName;
 org.apache.log4j.BasicConfigurator.configure();
    // Check arguments length value
    if(args.length == 0){
        topicName= "Donosti";
```

```
       System.out.println(topicName);
    }
    else {
        //Assign topicName to string variable
       topicName = args[0].toString();

    }

    // create instance for properties to access producer configs
    Properties props = new Properties();

    //Assign localhost id
    props.put("bootstrap.servers", "localhost:9092");

    //Set acknowledgements for producer requests.
    props.put("acks", "all");

    //If the request fails, the producer can automatically retry,
    props.put("retries", 0);

    //Specify buffer size in config
    props.put("batch.size", 16384);

    //Reduce the no of requests less than 0
    props.put("linger.ms", 1);

    //The buffer.memory controls the total amount of memory available
to the producer for buffering.
    props.put("buffer.memory", 33554432);

    props.put("key.serializer",
       "org.apache.kafka.common.serialization.StringSerializer");

    props.put("value.serializer",
       "org.apache.kafka.common.serialization.StringSerializer");

    Producer<String, String> producer = new KafkaProducer
       <String, String>(props);


    Timestamp ts = new Timestamp (System.currentTimeMillis());

    // para pruebas en lugar de estar produciendo indefinidamente, voy
a hacer que produzca 100 valores
    int times=0;
    for(int i = 0; i < 100; i++)
    {
        JSONObject windmeasures = new JSONObject();
        windmeasures.put("timestamp", new Timestamp
(System.currentTimeMillis()).getTime());
        windmeasures.put("location", topicName);
        windmeasures.put("velocity", Math.random() * 80);


       producer.send(new ProducerRecord<String,
String>(topicName,Integer.toString(i),windmeasures.toString()));
       //System.out.println(windmeasures);
       Thread.sleep(5000);
       times=i;
    }
```

```
            System.out.println("Message sent successfully "+times+"
times");
            producer.close();
}
}
```

MarkWrongBolt.java

```
package kafkastorm.ejemplo;

import java.sql.Timestamp;
import java.util.Map;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;


import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;
import kafka.utils.Json;
import backtype.storm.task.OutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.IRichBolt;
import backtype.storm.task.TopologyContext;

public class MarkWrongBolt implements IRichBolt {
   private OutputCollector collector;
   public int wrongmeasuresBilbao;
   public int wrongmeasuresDonosti;
   public int evaluatedBilbao;
   public int evaluatedDonosti;

   //@Override
   public void prepare(Map stormConf, TopologyContext context,
OutputCollector collector) {
      this.collector = collector;
   }

   //@Override
   public void execute(Tuple input) {
      JSONObject windbolt = null;
      //JSONObject windboltwrong = null;

      try {
          windbolt = (JSONObject) new
JSONParser().parse(input.getString(0));

      } catch (ParseException e) {
          e.printStackTrace();
      }


        Double velocity = (Double) windbolt.get("velocity");
        String city = (String) windbolt.get("location");
```

```
        if (city.equalsIgnoreCase("Donosti"))
        {
            evaluatedDonosti=evaluatedDonosti+1;

        }
    if (city.equalsIgnoreCase("Bilbao"))
    {
            evaluatedBilbao=evaluatedBilbao+1;

    }

        if (velocity > 50.00)
        {
            if (city.equalsIgnoreCase("Donosti"))
            {
                wrongmeasuresDonosti=wrongmeasuresDonosti+1;
                windbolt.put("wrong", wrongmeasuresDonosti);
                windbolt.put("total", evaluatedDonosti);
            }
        if (city.equalsIgnoreCase("Bilbao"))
        {
            wrongmeasuresBilbao=wrongmeasuresBilbao+1;
            windbolt.put("wrong", wrongmeasuresBilbao);
            windbolt.put("total", evaluatedBilbao);
        }

            }
        collector.emit(new Values(windbolt.toJSONString()));
      collector.ack(input);
   }

  // @Override
   public void declareOutputFields(OutputFieldsDeclarer declarer) {
       declarer.declare(new Fields("location"));
   }

   //@Override
   public void cleanup() {}

   //@Override
   public Map<String, Object> getComponentConfiguration() {
      return null;
   }

}
```

StoreBolt

```
package kafkastorm.ejemplo;

import java.util.Map;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
```

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.sql.Timestamp;
import java.util.HashMap;

import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;
import backtype.storm.task.OutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.IRichBolt;
import backtype.storm.task.TopologyContext;

public class StoreBolt implements IRichBolt{
   Map<String, Integer> counters;
   private OutputCollector collector;
  // public JSONObject windboltmax = new JSONObject();
   public int evaluatedDonosti;
   public int evaluatedBilbao;

  // @Override
   public void prepare(Map stormConf, TopologyContext context,
OutputCollector collector) {
      this.counters = new HashMap<String, Integer>();
      this.collector = collector;
   }

   //@Override
   public void execute(Tuple input) {
   //JSONParser parser = new JSONParser();

   JSONObject windbolt = null;

   try {
       windbolt = (JSONObject) new
JSONParser().parse(input.getString(0));

   } catch (ParseException e) {
       e.printStackTrace();
   }

   //escribo en el fichero

      String Ficheroloc = windbolt.get("location").toString();

      try {
      File archivo=new File("/home/osboxes/kafka-
storm/wind"+Ficheroloc+".txt");

      FileWriter escribir=new FileWriter(archivo,true);
      Object res = windbolt.get("wrong");
        //aÑAado el cálculo de latencia, tiempo transcurrido desde que
se emite el dato hasta que se almacena
        long currentTime = new Timestamp
(System.currentTimeMillis()).getTime();
        long datatime = (Long) windbolt.get("timestamp");
        long latency = (currentTime - datatime);
         if (windbolt.get("wrong") == null)
            {
```

```
            windbolt.put("latency", latency);
        escribir.write(windbolt.toString());
        collector.emit(new Values(windbolt.toJSONString()));
        // Escribimos linea a linea en el fichero
            }
         else {
            escribir.write("\n");
               escribir.write(windbolt.get("location")+": "+"Llevamos
"+windbolt.get("wrong")+" medidas mayores de 50, de entre:
"+windbolt.get("total")+" evaluadas.");
               escribir.write("\n");
               collector.emit(new Values(windbolt.get("location")+":
"+"Llevamos "+windbolt.get("wrong")+" medidas mayores de 50, de entre:
"+windbolt.get("total")+" evaluadas."));
         }
            escribir.close();
    } catch (Exception ex) {
        System.out.println("Mensaje de la excepciÃ³n: " +
ex.getMessage());
    }
      collector.ack(input);
   }

   //@Override
   public void cleanup() {
   }

   //@Override
   public void declareOutputFields(OutputFieldsDeclarer declarer) {
          declarer.declare(new Fields("default"));
   }

  // @Override
   public Map<String, Object> getComponentConfiguration() {
      return null;
   }
}
```

WindMeasures.java

```
package kafkastorm.ejemplo;
import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.tuple.Fields;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.UUID;

import backtype.storm.spout.SchemeAsMultiScheme;
import storm.kafka.trident.GlobalPartitionInformation;
import storm.kafka.ZkHosts;
import storm.kafka.Broker;
import storm.kafka.StaticHosts;
import storm.kafka.BrokerHosts;
import storm.kafka.SpoutConfig;
import storm.kafka.KafkaConfig;
```

```
import storm.kafka.KafkaSpout;
import storm.kafka.StringScheme;
//ejemplo hdfs
import backtype.storm.Config;
import backtype.storm.StormSubmitter;
import backtype.storm.topology.TopologyBuilder;

import backtype.storm.tuple.Fields;

import org.apache.storm.hdfs.bolt.HdfsBolt;
import org.apache.storm.hdfs.bolt.format.DefaultFileNameFormat;
import org.apache.storm.hdfs.bolt.format.DelimitedRecordFormat;
import org.apache.storm.hdfs.bolt.format.RecordFormat;
import org.apache.storm.hdfs.bolt.rotation.FileRotationPolicy;
import org.apache.storm.hdfs.bolt.rotation.FileSizeRotationPolicy;
import org.apache.storm.hdfs.bolt.sync.CountSyncPolicy;
import org.apache.storm.hdfs.bolt.sync.SyncPolicy;
//fin ejemplo hdfs

public class WindMeasures {
   public static void main(String[] args) throws Exception{
      Config config = new Config();
      config.setDebug(true);
      config.put(Config.TOPOLOGY_MAX_SPOUT_PENDING, 1);
      String zkConnString = "localhost:2181";
      BrokerHosts hosts = new ZkHosts(zkConnString);
      //uno va a ser para el spout donosti y otro para el bilbao
      //así, dependiendo de cuÃ¡l sea el topic asociado, construye el
dato jason con el nombre del topic como
      //propiedad location
      //para esto, el producer bilbao ni el donosti no incluyen
propiedad location
      System.out.println("Enter topic name:");
      //return;
      Scanner scanInput = new Scanner(System.in);
      String topic1= scanInput.nextLine();

      //scanInput.close();
      System.out.println(topic1);
      //String topic = "test-topic";
      System.out.println("Enter another topic name:");
      scanInput = new Scanner(System.in);
      String topic2= scanInput.nextLine();

      scanInput.close();
      System.out.println(topic2);
      //String topic = "test-topic";

      //ahora creo 2 spouts con la misma config pero con distintos
topic name
      SpoutConfig Spout1Config = new SpoutConfig (hosts, topic1, "/" +
topic1, UUID.randomUUID().toString());
      Spout1Config.bufferSizeBytes = 1024 * 1024 * 4;
      Spout1Config.fetchSizeBytes = 1024 * 1024 * 4;
      Spout1Config.forceFromStart = true;
      Spout1Config.scheme = new SchemeAsMultiScheme(new
StringScheme());

      SpoutConfig Spout2Config = new SpoutConfig (hosts, topic2, "/" +
topic2, UUID.randomUUID().toString());
      Spout2Config.bufferSizeBytes = 1024 * 1024 * 4;
```

```
      Spout2Config.fetchSizeBytes = 1024 * 1024 * 4;
      Spout2Config.forceFromStart = true;
      Spout2Config.scheme = new SchemeAsMultiScheme(new
StringScheme());

      //It is a comma separated value file (although only one
word per line is written)
      RecordFormat format = new
DelimitedRecordFormat().withFieldDelimiter(",");
      SyncPolicy syncPolicy = new CountSyncPolicy(1000);

      //Rotate files after 127MB (Hortonworks default fileblock
size is 128MB)
      FileRotationPolicy rotationPolicy = new
FileSizeRotationPolicy(127.0f, FileSizeRotationPolicy.Units.MB);

      DefaultFileNameFormat fileNameFormat = new
DefaultFileNameFormat();
      //The files are written in this HDFS folder
      fileNameFormat.withPath("/user/osboxes");
      //Files start with the following filename prefix
      fileNameFormat.withPrefix("wind");
      //Files end with the following suffix
      fileNameFormat.withExtension(".txt");

      //HDFS bolt
      HdfsBolt bolt =
          new HdfsBolt().withFsUrl("hdfs://localhost:9000")
              .withFileNameFormat(fileNameFormat)
              .withRecordFormat(format)
              .withRotationPolicy(rotationPolicy)
              .withSyncPolicy(syncPolicy);



      TopologyBuilder builder = new TopologyBuilder();
      builder.setSpout("spout-location1", new
KafkaSpout(Spout1Config));
      builder.setSpout("spout-location2", new
KafkaSpout(Spout2Config));
      builder.setBolt("wrong-measures", new
MarkWrongBolt()).shuffleGrouping("spout-
location1").shuffleGrouping("spout-location2");
      builder.setBolt("store", new
StoreBolt()).shuffleGrouping("wrong-measures");
builder.setBolt("hdfs-wind", bolt). shuffleGrouping ("store");

      LocalCluster cluster = new LocalCluster();
      cluster.submitTopology("WindMeasures", config,
builder.createTopology());

     // esperando alrededor de 15 minutos, ya que los producers emiten
datos durante algo menos.
      Thread.sleep(900000);

     cluster.shutdown();
```

```
    }
}
```

Pom.xml

```
<project                        xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>kafkastorm</groupId>
  <artifactId>ejemplo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>ejemplo</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.storm</groupId>
        <artifactId>storm-core</artifactId>
        <version>0.9.7</version>
    </dependency>
   <dependency>
        <groupId>org.apache.curator</groupId>
        <artifactId>curator-client</artifactId>
        <version>2.9.1</version>
    </dependency>
   <dependency>
        <groupId>org.apache.curator</groupId>
        <artifactId>curator-framework</artifactId>
        <version>2.9.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.storm</groupId>
        <artifactId>storm-kafka</artifactId>
        <version>0.9.5</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>0.8.2.2</version>
    </dependency>
    <dependency>
       <groupId>org.apache.kafka</groupId>
       <artifactId>kafka_2.11</artifactId>
       <version>0.8.2.2</version>
```

```
     </dependency>
         <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.2.2</version>
    </dependency>
    <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.2.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>2.2.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
    <!--     https://mvnrepository.com/artifact/org.apache.storm/storm-
hdfs -->
<dependency>
    <groupId>org.apache.storm</groupId>
    <artifactId>storm-hdfs</artifactId>
    <version>0.9.3</version>
</dependency>

  </dependencies>
   <build>
   <plugins>
   <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.2</version>
    <configuration>
      <source>1.7</source>
      <target>1.7</target>
    </configuration>
  </plugin>
    <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>1.4</version>
    <configuration>
        <createDependencyReducedPom>true</createDependencyReducedPom>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>shade</goal>
            </goals>
            <configuration>
```

```
                <transformers>
                    <transformer

implementation="org.apache.maven.plugins.shade.resource.ServicesResour
ceTransformer"/>
                    <transformer

implementation="org.apache.maven.plugins.shade.resource.ManifestResour
ceTransformer">
                        <mainClass></mainClass>
                    </transformer>
                </transformers>
            </configuration>
        </execution>
    </executions>
</plugin>
</plugins>
 </build>
</project>
```

ANEXO III: Resultados.

windBilbao.txt:


```
{"latency":37059,"location":"Bilbao","velocity":29.51075086877463,"tim
estamp":1496478183457}
Bilbao: Llevamos 1 medidas mayores de 50, de entre: 2 evaluadas.
{"latency":26887,"location":"Bilbao","velocity":0.9864651307955974,"ti
mestamp":1496478193749}
Bilbao: Llevamos 2 medidas mayores de 50, de entre: 4 evaluadas.

Bilbao: Llevamos 3 medidas mayores de 50, de entre: 5 evaluadas.
{"latency":11971,"location":"Bilbao","velocity":8.413894073625965,"tim
estamp":1496478208751}{"latency":6989,"location":"Bilbao","velocity":2
9.076066952827617,"timestamp":1496478213751}
Bilbao: Llevamos 4 medidas mayores de 50, de entre: 8 evaluadas.

Bilbao: Llevamos 5 medidas mayores de 50, de entre: 9 evaluadas.

Bilbao: Llevamos 6 medidas mayores de 50, de entre: 10 evaluadas.

Bilbao: Llevamos 7 medidas mayores de 50, de entre: 11 evaluadas.

Bilbao: Llevamos 8 medidas mayores de 50, de entre: 12 evaluadas.

Bilbao: Llevamos 9 medidas mayores de 50, de entre: 13 evaluadas.
{"latency":6,"location":"Bilbao","velocity":49.38855484495921,"timesta
mp":1496478248753}{"latency":11,"location":"Bilbao","velocity":30.5533
43295328972,"timestamp":1496478253754}
Bilbao: Llevamos 10 medidas mayores de 50, de entre: 16 evaluadas.
{"latency":11,"location":"Bilbao","velocity":44.138121797257895,"times
tamp":1496478263754}{"latency":5,"location":"Bilbao","velocity":4.1887
60725755918,"timestamp":1496478268755}
Bilbao: Llevamos 11 medidas mayores de 50, de entre: 19 evaluadas.

Bilbao: Llevamos 12 medidas mayores de 50, de entre: 20 evaluadas.

Bilbao: Llevamos 13 medidas mayores de 50, de entre: 21 evaluadas.
{"latency":9,"location":"Bilbao","velocity":9.006228803215306,"timesta
mp":1496478288756}{"latency":6,"location":"Bilbao","velocity":21.80404
5506499612,"timestamp":1496478293756}
Bilbao: Llevamos 14 medidas mayores de 50, de entre: 24 evaluadas.

Bilbao: Llevamos 15 medidas mayores de 50, de entre: 25 evaluadas.

Bilbao: Llevamos 16 medidas mayores de 50, de entre: 26 evaluadas.

Bilbao: Llevamos 17 medidas mayores de 50, de entre: 27 evaluadas.
{"latency":7,"location":"Bilbao","velocity":6.6721831391104125,"timest
amp":1496478318757}{"latency":4,"location":"Bilbao","velocity":8.89589
7994792774,"timestamp":1496478323758}{"latency":8,"location":"Bilbao",
"velocity":32.996002013703595,"timestamp":1496478328758}
Bilbao: Llevamos 18 medidas mayores de 50, de entre: 31 evaluadas.
{"latency":5,"location":"Bilbao","velocity":7.95668376483734,"timestam
p":1496478338759}
Bilbao: Llevamos 19 medidas mayores de 50, de entre: 33 evaluadas.
{"latency":5,"location":"Bilbao","velocity":21.640952235412698,"timest
amp":1496478348760}{"latency":5,"location":"Bilbao","velocity":20.5836
74947600336,"timestamp":1496478353760}
Bilbao: Llevamos 20 medidas mayores de 50, de entre: 36 evaluadas.
```

Bilbao: Llevamos 21 medidas mayores de 50, de entre: 37 evaluadas.

Bilbao: Llevamos 22 medidas mayores de 50, de entre: 38 evaluadas.

Bilbao: Llevamos 23 medidas mayores de 50, de entre: 39 evaluadas.
{"latency":6,"location":"Bilbao","velocity":7.033352825677532,"timesta
mp":1496478378791}
Bilbao: Llevamos 24 medidas mayores de 50, de entre: 41 evaluadas.

Bilbao: Llevamos 25 medidas mayores de 50, de entre: 42 evaluadas.
{"latency":12,"location":"Bilbao","velocity":32.62962566922238,"timest
amp":1496478393793}
Bilbao: Llevamos 26 medidas mayores de 50, de entre: 44 evaluadas.

Bilbao: Llevamos 27 medidas mayores de 50, de entre: 45 evaluadas.
{"latency":8,"location":"Bilbao","velocity":48.187510775114745,"timest
amp":1496478408794}{"latency":4,"location":"Bilbao","velocity":48.3702
8289084769,"timestamp":1496478413794}
Bilbao: Llevamos 28 medidas mayores de 50, de entre: 48 evaluadas.
{"latency":8,"location":"Bilbao","velocity":12.062963686331365,"timest
amp":1496478423795}{"latency":5,"location":"Bilbao","velocity":22.3700
67932197443,"timestamp":1496478428795}{"latency":4,"location":"Bilbao"
,"velocity":24.43495955954269,"timestamp":1496478433795}{"latency":4,"
location":"Bilbao","velocity":30.217170430745448,"timestamp":149647843
8796}{"latency":8,"location":"Bilbao","velocity":5.621592535916999,"ti
mestamp":1496478443796}{"latency":5,"location":"Bilbao","velocity":37.
714226933291116,"timestamp":1496478448796}
Bilbao: Llevamos 29 medidas mayores de 50, de entre: 55 evaluadas.

Bilbao: Llevamos 30 medidas mayores de 50, de entre: 56 evaluadas.

Bilbao: Llevamos 31 medidas mayores de 50, de entre: 57 evaluadas.
{"latency":4,"location":"Bilbao","velocity":25.062963175298066,"timest
amp":1496478468798}{"latency":7,"location":"Bilbao","velocity":2.09922
0453951167,"timestamp":1496478473798}{"latency":5,"location":"Bilbao",
"velocity":18.6941343119042,"timestamp":1496478478798}{"latency":11,"l
ocation":"Bilbao","velocity":45.62993935359383,"timestamp":14964784837
98}{"latency":9,"location":"Bilbao","velocity":15.67282676583563,"time
stamp":1496478488799}
Bilbao: Llevamos 32 medidas mayores de 50, de entre: 63 evaluadas.

Bilbao: Llevamos 33 medidas mayores de 50, de entre: 64 evaluadas.
{"latency":8,"location":"Bilbao","velocity":2.8683977879958977,"timest
amp":1496478503800}{"latency":7,"location":"Bilbao","velocity":17.5712
08770115536,"timestamp":1496478508800}{"latency":27,"location":"Bilbao
","velocity":49.1467502687061,"timestamp":1496478513801}{"latency":11,
"location":"Bilbao","velocity":45.281078806451845,"timestamp":14964785
18801}{"latency":9,"location":"Bilbao","velocity":32.03056946970434,"t
imestamp":1496478523802}
Bilbao: Llevamos 34 medidas mayores de 50, de entre: 70 evaluadas.
{"latency":10,"location":"Bilbao","velocity":5.5087745527119125,"times
tamp":1496478533802}{"latency":6,"location":"Bilbao","velocity":8.1326
2114119115,"timestamp":1496478538803}{"latency":27,"location":"Bilbao"
,"velocity":41.87234141837692,"timestamp":1496478543803}{"latency":6,"
location":"Bilbao","velocity":17.25468465275786,"timestamp":1496478548
803}
Bilbao: Llevamos 35 medidas mayores de 50, de entre: 75 evaluadas.

Bilbao: Llevamos 36 medidas mayores de 50, de entre: 76 evaluadas.

{"latency":9,"location":"Bilbao","velocity":4.662556279889172,"timesta
mp":1496478563804}{"latency":7,"location":"Bilbao","velocity":1.048344
4767777872,"timestamp":1496478568805}
Bilbao: Llevamos 37 medidas mayores de 50, de entre: 79 evaluadas.
{"latency":5,"location":"Bilbao","velocity":28.69896767405752,"timesta
mp":1496478578806}{"latency":8,"location":"Bilbao","velocity":29.71822
774471857,"timestamp":1496478583806}{"latency":4,"location":"Bilbao","
velocity":41.99860113577196,"timestamp":1496478588807}{"latency":4,"lo
cation":"Bilbao","velocity":25.676145120206684,"timestamp":14964785938
07}
Bilbao: Llevamos 38 medidas mayores de 50, de entre: 84 evaluadas.

Bilbao: Llevamos 39 medidas mayores de 50, de entre: 85 evaluadas.
{"latency":5,"location":"Bilbao","velocity":28.081836530262123,"timest
amp":1496478608808}{"latency":4,"location":"Bilbao","velocity":19.5640
3077500906,"timestamp":1496478613808}
Bilbao: Llevamos 40 medidas mayores de 50, de entre: 88 evaluadas.
{"latency":4,"location":"Bilbao","velocity":15.177028619096848,"timest
amp":1496478623809}{"latency":5,"location":"Bilbao","velocity":4.36763
267380023,"timestamp":1496478628809}{"latency":6,"location":"Bilbao","
velocity":21.833178714331698,"timestamp":1496478633809}{"latency":5,"l
ocation":"Bilbao","velocity":15.126161149572237,"timestamp":1496478638
810}{"latency":13,"location":"Bilbao","velocity":17.974725807812533,"t
imestamp":1496478643810}{"latency":7,"location":"Bilbao","velocity":46
.166737694780274,"timestamp":1496478648810}
Bilbao: Llevamos 41 medidas mayores de 50, de entre: 95 evaluadas.

Bilbao: Llevamos 42 medidas mayores de 50, de entre: 96 evaluadas.

Bilbao: Llevamos 43 medidas mayores de 50, de entre: 97 evaluadas.
{"latency":5,"location":"Bilbao","velocity":18.538361515944565,"timest
amp":1496478668812}{"latency":4,"location":"Bilbao","velocity":36.4551
7175802192,"timestamp":1496478673812}{"latency":8,"location":"Bilbao",
"velocity":27.50728806633404,"timestamp":1496478678812}

windDonosti.txt:

{"latency":34606,"location":"Donosti","velocity":2.7529041291849854,"t
imestamp":1496478185917}{"latency":29573,"location":"Donosti","velocit
y":6.175384196836422,"timestamp":1496478191044}{"latency":24595,"locat
ion":"Donosti","velocity":4.780197637513552,"timestamp":1496478196045}
{"latency":19640,"location":"Donosti","velocity":18.07892642537338,"ti
mestamp":1496478201045}{"latency":14667,"location":"Donosti","velocity
":40.36428605353126,"timestamp":1496478206046}{"latency":9683,"locatio
n":"Donosti","velocity":10.558281687257924,"timestamp":1496478211046}{
"latency":4703,"location":"Donosti","velocity":2.0771642210531738,"tim
estamp":1496478216046}{"latency":15,"location":"Donosti","velocity":15
.695963205948491,"timestamp":1496478221047}{"latency":10,"location":"D
onosti","velocity":21.741881699595478,"timestamp":1496478226048}
Donosti: Llevamos 1 medidas mayores de 50, de entre: 10 evaluadas.
{"latency":14,"location":"Donosti","velocity":35.13546233761859,"times
tamp":1496478236050}
Donosti: Llevamos 2 medidas mayores de 50, de entre: 12 evaluadas.
{"latency":6,"location":"Donosti","velocity":19.04115022463378,"timest
amp":1496478246051}
Donosti: Llevamos 3 medidas mayores de 50, de entre: 14 evaluadas.
{"latency":7,"location":"Donosti","velocity":9.422699304724533,"timest
amp":1496478256051}{"latency":5,"location":"Donosti","velocity":33.400
49954546518,"timestamp":1496478261052}{"latency":13,"location":"Donost

i","velocity":38.2511454743712,"timestamp":1496478266052}{"latency":4,
"location":"Donosti","velocity":0.4596476939332472,"timestamp":1496478
271052}{"latency":5,"location":"Donosti","velocity":40.94094996008745,
"timestamp":1496478276053}{"latency":9,"location":"Donosti","velocity"
:46.4666827549792,"timestamp":1496478281053}
Donosti: Llevamos 4 medidas mayores de 50, de entre: 21 evaluadas.
{"latency":4,"location":"Donosti","velocity":14.591693852007781,"times
tamp":1496478291053}{"latency":6,"location":"Donosti","velocity":13.08
1159699273197,"timestamp":1496478296054}{"latency":10,"location":"Dono
sti","velocity":28.940226265674028,"timestamp":1496478301054}{"latency
":6,"location":"Donosti","velocity":41.542113934381746,"timestamp":149
6478306054}
Donosti: Llevamos 5 medidas mayores de 50, de entre: 26 evaluadas.

Donosti: Llevamos 6 medidas mayores de 50, de entre: 27 evaluadas.
{"latency":10,"location":"Donosti","velocity":41.31394861170616,"times
tamp":1496478321055}{"latency":5,"location":"Donosti","velocity":15.61
4005955427857,"timestamp":1496478326055}{"latency":6,"location":"Donos
ti","velocity":45.403566947540966,"timestamp":1496478331056}{"latency"
:6,"location":"Donosti","velocity":0.8363041260410409,"timestamp":1496
478336056}{"latency":10,"location":"Donosti","velocity":8.653922117678
832,"timestamp":1496478341057}
Donosti: Llevamos 7 medidas mayores de 50, de entre: 33 evaluadas.
{"latency":5,"location":"Donosti","velocity":39.881070983505474,"times
tamp":1496478351058}
Donosti: Llevamos 8 medidas mayores de 50, de entre: 35 evaluadas.

Donosti: Llevamos 9 medidas mayores de 50, de entre: 36 evaluadas.
{"latency":7,"location":"Donosti","velocity":42.14101882534735,"timest
amp":1496478366059}
Donosti: Llevamos 10 medidas mayores de 50, de entre: 38 evaluadas.

Donosti: Llevamos 11 medidas mayores de 50, de entre: 39 evaluadas.
{"latency":17,"location":"Donosti","velocity":12.853929461979599,"time
stamp":1496478381071}
Donosti: Llevamos 12 medidas mayores de 50, de entre: 41 evaluadas.
{"latency":5,"location":"Donosti","velocity":41.584547304385026,"times
tamp":1496478391072}{"latency":21,"location":"Donosti","velocity":15.8
68873679173703,"timestamp":1496478396072}{"latency":5,"location":"Dono
sti","velocity":19.75828108019284,"timestamp":1496478401072}{"latency"
:5,"location":"Donosti","velocity":17.370751345741667,"timestamp":1496
478406073}{"latency":5,"location":"Donosti","velocity":4.7194608042593
61,"timestamp":1496478411073}
Donosti: Llevamos 13 medidas mayores de 50, de entre: 47 evaluadas.
{"latency":8,"location":"Donosti","velocity":38.660087371616584,"times
tamp":1496478421074}{"latency":4,"location":"Donosti","velocity":16.26
3533493198665,"timestamp":1496478426074}{"latency":5,"location":"Donos
ti","velocity":28.48438075025505,"timestamp":1496478431075}
Donosti: Llevamos 14 medidas mayores de 50, de entre: 51 evaluadas.
{"latency":6,"location":"Donosti","velocity":22.752976614363565,"times
tamp":1496478441075}{"latency":6,"location":"Donosti","velocity":23.70
4382524771155,"timestamp":1496478446075}
Donosti: Llevamos 15 medidas mayores de 50, de entre: 54 evaluadas.
{"latency":8,"location":"Donosti","velocity":15.204793523290316,"times
tamp":1496478456076}{"latency":10,"location":"Donosti","velocity":15.1
82308179807542,"timestamp":1496478461076}{"latency":8,"location":"Dono
sti","velocity":24.938843701096562,"timestamp":1496478466077}{"latency
":6,"location":"Donosti","velocity":39.93191673021975,"timestamp":1496
478471077}
Donosti: Llevamos 16 medidas mayores de 50, de entre: 59 evaluadas.

{"latency":4,"location":"Donosti","velocity":37.39464689184501,"timest
amp":1496478481078}
Donosti: Llevamos 17 medidas mayores de 50, de entre: 61 evaluadas.

Donosti: Llevamos 18 medidas mayores de 50, de entre: 62 evaluadas.
{"latency":6,"location":"Donosti","velocity":19.510314513253057,"times
tamp":1496478496079}
Donosti: Llevamos 19 medidas mayores de 50, de entre: 64 evaluadas.
{"latency":6,"location":"Donosti","velocity":24.37119340527591,"timest
amp":1496478506079}{"latency":7,"location":"Donosti","velocity":0.8166
35195633042,"timestamp":1496478511080}
Donosti: Llevamos 20 medidas mayores de 50, de entre: 67 evaluadas.
{"latency":4,"location":"Donosti","velocity":42.86715297004707,"timest
amp":1496478521080}{"latency":7,"location":"Donosti","velocity":7.5158
19058231967,"timestamp":1496478526080}{"latency":5,"location":"Donosti
","velocity":9.601523969978265,"timestamp":1496478531081}{"latency":5,
"location":"Donosti","velocity":32.47330544351587,"timestamp":14964785
36081}{"latency":7,"location":"Donosti","velocity":16.80080177811422,"
timestamp":1496478541081}
Donosti: Llevamos 21 medidas mayores de 50, de entre: 73 evaluadas.
{"latency":12,"location":"Donosti","velocity":30.5976108672835,"timest
amp":1496478551082}{"latency":5,"location":"Donosti","velocity":45.271
856714049086,"timestamp":1496478556082}{"latency":5,"location":"Donost
i","velocity":31.87690614850031,"timestamp":1496478561083}{"latency":5
,"location":"Donosti","velocity":17.18168635361365,"timestamp":1496478
566083}{"latency":8,"location":"Donosti","velocity":40.541651114059334
,"timestamp":1496478571083}{"latency":8,"location":"Donosti","velocity
":26.935879216576282,"timestamp":1496478576084}
Donosti: Llevamos 22 medidas mayores de 50, de entre: 80 evaluadas.
{"latency":4,"location":"Donosti","velocity":8.76332237253636,"timesta
mp":1496478586084}
Donosti: Llevamos 23 medidas mayores de 50, de entre: 82 evaluadas.
{"latency":5,"location":"Donosti","velocity":6.735454639058656,"timest
amp":1496478596085}{"latency":12,"location":"Donosti","velocity":43.70
408991530516,"timestamp":1496478601085}
Donosti: Llevamos 24 medidas mayores de 50, de entre: 85 evaluadas.
{"latency":4,"location":"Donosti","velocity":15.973606433668444,"times
tamp":1496478611086}{"latency":4,"location":"Donosti","velocity":49.03
281774884797,"timestamp":1496478616086}{"latency":12,"location":"Donos
ti","velocity":35.29172035636388,"timestamp":1496478621087}{"latency":
4,"location":"Donosti","velocity":15.111714325490553,"timestamp":14964
78626087}{"latency":5,"location":"Donosti","velocity":14.3335532108221
56,"timestamp":1496478631087}{"latency":8,"location":"Donosti","veloci
ty":20.40600482058542,"timestamp":1496478636087}
Donosti: Llevamos 25 medidas mayores de 50, de entre: 92 evaluadas.
{"latency":8,"location":"Donosti","velocity":10.299903692506351,"times
tamp":1496478646088}{"latency":13,"location":"Donosti","velocity":34.5
501652129124 25,"timestamp":1496478651088}{"latency":5,"location":"Dono
sti","velocity":26.33007036953284,"timestamp":1496478656088}
Donosti: Llevamos 26 medidas mayores de 50, de entre: 96 evaluadas.

Donosti: Llevamos 27 medidas mayores de 50, de entre: 97 evaluadas.

Donosti: Llevamos 28 medidas mayores de 50, de entre: 98 evaluadas.
{"latency":9,"location":"Donosti","velocity":39.64487381240855,"timest
amp":1496478676089}{"latency":7,"location":"Donosti","velocity":49.454
386399618514,"timestamp":1496478681090}

windhdfs-wind-2-0-1496427282306.txt

{"latency":37059,"location":"Bilbao","velocity":29.51075086877463,"tim
estamp":1496478183457}
{"latency":34606,"location":"Donosti","velocity":2.7529041291849854,"t
imestamp":1496478185917}
Bilbao: Llevamos 1 medidas mayores de 50, de entre: 2 evaluadas.
{"latency":29573,"location":"Donosti","velocity":6.175384196836422,"ti
mestamp":1496478191044}
{"latency":26887,"location":"Bilbao","velocity":0.9864651307955974,"ti
mestamp":1496478193749}
{"latency":24595,"location":"Donosti","velocity":4.780197637513552,"ti
mestamp":1496478196045}
Bilbao: Llevamos 2 medidas mayores de 50, de entre: 4 evaluadas.
{"latency":19640,"location":"Donosti","velocity":18.07892642537338,"ti
mestamp":1496478201045}
Bilbao: Llevamos 3 medidas mayores de 50, de entre: 5 evaluadas.
{"latency":14667,"location":"Donosti","velocity":40.36428605353126,"ti
mestamp":1496478206046}
{"latency":11971,"location":"Bilbao","velocity":8.413894073625965,"tim
estamp":1496478208751}
{"latency":9683,"location":"Donosti","velocity":10.558281687257924,"ti
mestamp":1496478211046}
{"latency":6989,"location":"Bilbao","velocity":29.076066952827617,"tim
estamp":1496478213751}
{"latency":4703,"location":"Donosti","velocity":2.0771642210531738,"ti
mestamp":1496478216046}
Bilbao: Llevamos 4 medidas mayores de 50, de entre: 8 evaluadas.
{"latency":15,"location":"Donosti","velocity":15.695963205948491,"time
stamp":1496478221047}
Bilbao: Llevamos 5 medidas mayores de 50, de entre: 9 evaluadas.
{"latency":10,"location":"Donosti","velocity":21.741881699595478,"time
stamp":1496478226048}
Bilbao: Llevamos 6 medidas mayores de 50, de entre: 10 evaluadas.
Donosti: Llevamos 1 medidas mayores de 50, de entre: 10 evaluadas.
Bilbao: Llevamos 7 medidas mayores de 50, de entre: 11 evaluadas.
{"latency":14,"location":"Donosti","velocity":35.13546233761859,"times
tamp":1496478236050}
Bilbao: Llevamos 8 medidas mayores de 50, de entre: 12 evaluadas.
Donosti: Llevamos 2 medidas mayores de 50, de entre: 12 evaluadas.
Bilbao: Llevamos 9 medidas mayores de 50, de entre: 13 evaluadas.
{"latency":6,"location":"Donosti","velocity":19.04115022463378,"timest
amp":1496478246051}
{"latency":6,"location":"Bilbao","velocity":49.38855484495921,"timesta
mp":1496478248753}
Donosti: Llevamos 3 medidas mayores de 50, de entre: 14 evaluadas.
{"latency":11,"location":"Bilbao","velocity":30.553343295328972,"times
tamp":1496478253754}
{"latency":7,"location":"Donosti","velocity":9.422699304724533,"timest
amp":1496478256051}
Bilbao: Llevamos 10 medidas mayores de 50, de entre: 16 evaluadas.
{"latency":5,"location":"Donosti","velocity":33.40049954546518,"timest
amp":1496478261052}
{"latency":11,"location":"Bilbao","velocity":44.138121797257895,"times
tamp":1496478263754}
{"latency":13,"location":"Donosti","velocity":38.2511454743712,"timest
amp":1496478266052}
{"latency":5,"location":"Bilbao","velocity":4.188760725755918,"timesta
mp":1496478268755}
{"latency":4,"location":"Donosti","velocity":0.4596476939332472,"times
tamp":1496478271052}

```
Bilbao: Llevamos 11 medidas mayores de 50, de entre: 19 evaluadas.
{"latency":5,"location":"Donosti","velocity":40.94094996008745,"timest
amp":1496478276053}
Bilbao: Llevamos 12 medidas mayores de 50, de entre: 20 evaluadas.
{"latency":9,"location":"Donosti","velocity":46.4666827549792,"timesta
mp":1496478281053}
Bilbao: Llevamos 13 medidas mayores de 50, de entre: 21 evaluadas.
Donosti: Llevamos 4 medidas mayores de 50, de entre: 21 evaluadas.
{"latency":9,"location":"Bilbao","velocity":9.006228803215306,"timesta
mp":1496478288756}
{"latency":4,"location":"Donosti","velocity":14.591693852007781,"times
tamp":1496478291053}
{"latency":6,"location":"Bilbao","velocity":21.804045506499612,"timest
amp":1496478293756}
{"latency":6,"location":"Donosti","velocity":13.081159699273197,"times
tamp":1496478296054}
Bilbao: Llevamos 14 medidas mayores de 50, de entre: 24 evaluadas.
{"latency":10,"location":"Donosti","velocity":28.940226265674028,"time
stamp":1496478301054}
Bilbao: Llevamos 15 medidas mayores de 50, de entre: 25 evaluadas.
{"latency":6,"location":"Donosti","velocity":41.542113934381746,"times
tamp":1496478306054}
Bilbao: Llevamos 16 medidas mayores de 50, de entre: 26 evaluadas.
Donosti: Llevamos 5 medidas mayores de 50, de entre: 26 evaluadas.
Bilbao: Llevamos 17 medidas mayores de 50, de entre: 27 evaluadas.
Donosti: Llevamos 6 medidas mayores de 50, de entre: 27 evaluadas.
{"latency":7,"location":"Bilbao","velocity":6.6721831391104125,"timest
amp":1496478318757}
{"latency":10,"location":"Donosti","velocity":41.31394861170616,"times
tamp":1496478321055}
{"latency":4,"location":"Bilbao","velocity":8.895897994792774,"timesta
mp":1496478323758}
{"latency":5,"location":"Donosti","velocity":15.614005955427857,"times
tamp":1496478326055}
{"latency":8,"location":"Bilbao","velocity":32.996002013703595,"timest
amp":1496478328758}
{"latency":6,"location":"Donosti","velocity":45.403566947540966,"times
tamp":1496478331056}
Bilbao: Llevamos 18 medidas mayores de 50, de entre: 31 evaluadas.
{"latency":6,"location":"Donosti","velocity":0.8363041260410409,"times
tamp":1496478336056}
{"latency":5,"location":"Bilbao","velocity":7.95668376483734,"timestam
p":1496478338759}
{"latency":10,"location":"Donosti","velocity":8.653922117678832,"times
tamp":1496478341057}
Bilbao: Llevamos 19 medidas mayores de 50, de entre: 33 evaluadas.
Donosti: Llevamos 7 medidas mayores de 50, de entre: 33 evaluadas.
{"latency":5,"location":"Bilbao","velocity":21.640952235412698,"timest
amp":1496478348760}
{"latency":5,"location":"Donosti","velocity":39.881070983505474,"times
tamp":1496478351058}
{"latency":5,"location":"Bilbao","velocity":20.583674947600336,"timest
amp":1496478353760}
Donosti: Llevamos 8 medidas mayores de 50, de entre: 35 evaluadas.
Bilbao: Llevamos 20 medidas mayores de 50, de entre: 36 evaluadas.
Donosti: Llevamos 9 medidas mayores de 50, de entre: 36 evaluadas.
Bilbao: Llevamos 21 medidas mayores de 50, de entre: 37 evaluadas.
{"latency":7,"location":"Donosti","velocity":42.14101882534735,"timest
amp":1496478366059}
Bilbao: Llevamos 22 medidas mayores de 50, de entre: 38 evaluadas.
Donosti: Llevamos 10 medidas mayores de 50, de entre: 38 evaluadas.
```

```
Bilbao: Llevamos 23 medidas mayores de 50, de entre: 39 evaluadas.
Donosti: Llevamos 11 medidas mayores de 50, de entre: 39 evaluadas.
{"latency":6,"location":"Bilbao","velocity":7.033352825677532,"timesta
mp":1496478378791}
{"latency":17,"location":"Donosti","velocity":12.853929461979599,"time
stamp":1496478381071}
Bilbao: Llevamos 24 medidas mayores de 50, de entre: 41 evaluadas.
Donosti: Llevamos 12 medidas mayores de 50, de entre: 41 evaluadas.
Bilbao: Llevamos 25 medidas mayores de 50, de entre: 42 evaluadas.
{"latency":5,"location":"Donosti","velocity":41.584547304385026,"times
tamp":1496478391072}
{"latency":12,"location":"Bilbao","velocity":32.62962566922238,"timest
amp":1496478393793}
{"latency":21,"location":"Donosti","velocity":15.868873679173703,"time
stamp":1496478396072}
Bilbao: Llevamos 26 medidas mayores de 50, de entre: 44 evaluadas.
{"latency":5,"location":"Donosti","velocity":19.75828108019284,"timest
amp":1496478401072}
Bilbao: Llevamos 27 medidas mayores de 50, de entre: 45 evaluadas.
{"latency":5,"location":"Donosti","velocity":17.370751345741667,"times
tamp":1496478406073}
{"latency":8,"location":"Bilbao","velocity":48.187510775114745,"timest
amp":1496478408794}
{"latency":5,"location":"Donosti","velocity":4.719460804259361,"timest
amp":1496478411073}
{"latency":4,"location":"Bilbao","velocity":48.37028289084769,"timesta
mp":1496478413794}
Donosti: Llevamos 13 medidas mayores de 50, de entre: 47 evaluadas.
Bilbao: Llevamos 28 medidas mayores de 50, de entre: 48 evaluadas.
{"latency":8,"location":"Donosti","velocity":38.660087371616584,"times
tamp":1496478421074}
{"latency":8,"location":"Bilbao","velocity":12.062963686331365,"timest
amp":1496478423795}
{"latency":4,"location":"Donosti","velocity":16.263533493198665,"times
tamp":1496478426074}
{"latency":5,"location":"Bilbao","velocity":22.370067932197443,"timest
amp":1496478428795}
{"latency":5,"location":"Donosti","velocity":28.48438075025505,"timest
amp":1496478431075}
{"latency":4,"location":"Bilbao","velocity":24.43495955954269,"timesta
mp":1496478433795}
Donosti: Llevamos 14 medidas mayores de 50, de entre: 51 evaluadas.
{"latency":4,"location":"Bilbao","velocity":30.217170430745448,"timest
amp":1496478438796}
{"latency":6,"location":"Donosti","velocity":22.752976614363565,"times
tamp":1496478441075}
{"latency":8,"location":"Bilbao","velocity":5.621592535916999,"timesta
mp":1496478443796}
{"latency":6,"location":"Donosti","velocity":23.704382524771155,"times
tamp":1496478446075}
{"latency":5,"location":"Bilbao","velocity":37.714226933291116,"timest
amp":1496478448796}
Donosti: Llevamos 15 medidas mayores de 50, de entre: 54 evaluadas.
Bilbao: Llevamos 29 medidas mayores de 50, de entre: 55 evaluadas.
{"latency":8,"location":"Donosti","velocity":15.204793523290316,"times
tamp":1496478456076}
Bilbao: Llevamos 30 medidas mayores de 50, de entre: 56 evaluadas.
{"latency":10,"location":"Donosti","velocity":15.182308179807542,"time
stamp":1496478461076}
Bilbao: Llevamos 31 medidas mayores de 50, de entre: 57 evaluadas.
```

{"latency":8,"location":"Donosti","velocity":24.938843701096562,"timestamp":1496478466077}
{"latency":4,"location":"Bilbao","velocity":25.062963175298066,"timestamp":1496478468798}
{"latency":6,"location":"Donosti","velocity":39.93191673021975,"timestamp":1496478471077}
{"latency":7,"location":"Bilbao","velocity":2.099220453951167,"timestamp":1496478473798}
Donosti: Llevamos 16 medidas mayores de 50, de entre: 59 evaluadas.
{"latency":5,"location":"Bilbao","velocity":18.6941343119042,"timestamp":1496478478798}
{"latency":4,"location":"Donosti","velocity":37.39464689184501,"timestamp":1496478481078}
{"latency":11,"location":"Bilbao","velocity":45.62993935359383,"timestamp":1496478483798}
Donosti: Llevamos 17 medidas mayores de 50, de entre: 61 evaluadas.
{"latency":9,"location":"Bilbao","velocity":15.67282676583563,"timestamp":1496478488799}
Donosti: Llevamos 18 medidas mayores de 50, de entre: 62 evaluadas.
Bilbao: Llevamos 32 medidas mayores de 50, de entre: 63 evaluadas.
{"latency":6,"location":"Donosti","velocity":19.510314513253057,"timestamp":1496478496079}
Bilbao: Llevamos 33 medidas mayores de 50, de entre: 64 evaluadas.
Donosti: Llevamos 19 medidas mayores de 50, de entre: 64 evaluadas.
{"latency":8,"location":"Bilbao","velocity":2.8683977879958977,"timestamp":1496478503800}
{"latency":6,"location":"Donosti","velocity":24.37119340527591,"timestamp":1496478506079}
{"latency":7,"location":"Bilbao","velocity":17.571208770115536,"timestamp":1496478508800}
{"latency":7,"location":"Donosti","velocity":0.816635195633042,"timestamp":1496478511080}
{"latency":27,"location":"Bilbao","velocity":49.1467502687061,"timestamp":1496478513801}
Donosti: Llevamos 20 medidas mayores de 50, de entre: 67 evaluadas.
{"latency":11,"location":"Bilbao","velocity":45.281078806451845,"timestamp":1496478518801}
{"latency":4,"location":"Donosti","velocity":42.86715297004707,"timestamp":1496478521080}
{"latency":9,"location":"Bilbao","velocity":32.03056946970434,"timestamp":1496478523802}
{"latency":7,"location":"Donosti","velocity":7.515819058231967,"timestamp":1496478526080}
Bilbao: Llevamos 34 medidas mayores de 50, de entre: 70 evaluadas.
{"latency":5,"location":"Donosti","velocity":9.601523969978265,"timestamp":1496478531081}
{"latency":10,"location":"Bilbao","velocity":5.5087745527119125,"timestamp":1496478533802}
{"latency":5,"location":"Donosti","velocity":32.47330544351587,"timestamp":1496478536081}
{"latency":6,"location":"Bilbao","velocity":8.13262114119115,"timestamp":1496478538803}
{"latency":7,"location":"Donosti","velocity":16.80080177811422,"timestamp":1496478541081}
{"latency":27,"location":"Bilbao","velocity":41.87234141837692,"timestamp":1496478543803}
Donosti: Llevamos 21 medidas mayores de 50, de entre: 73 evaluadas.
{"latency":6,"location":"Bilbao","velocity":17.25468465275786,"timestamp":1496478548803}
{"latency":12,"location":"Donosti","velocity":30.5976108672835,"timestamp":1496478551082}

Bilbao: Llevamos 35 medidas mayores de 50, de entre: 75 evaluadas.
{"latency":5,"location":"Donosti","velocity":45.271856714049086,"times
tamp":1496478556082}
Bilbao: Llevamos 36 medidas mayores de 50, de entre: 76 evaluadas.
{"latency":5,"location":"Donosti","velocity":31.87690614850031,"timest
amp":1496478561083}
{"latency":9,"location":"Bilbao","velocity":4.662556279889172,"timesta
mp":1496478563804}
{"latency":5,"location":"Donosti","velocity":17.18168635361365,"timest
amp":1496478566083}
{"latency":7,"location":"Bilbao","velocity":1.0483444767777872,"timest
amp":1496478568805}
{"latency":8,"location":"Donosti","velocity":40.541651114059334,"times
tamp":1496478571083}
Bilbao: Llevamos 37 medidas mayores de 50, de entre: 79 evaluadas.
{"latency":8,"location":"Donosti","velocity":26.935879216576282,"times
tamp":1496478576084}
{"latency":5,"location":"Bilbao","velocity":28.69896767405752,"timesta
mp":1496478578806}
Donosti: Llevamos 22 medidas mayores de 50, de entre: 80 evaluadas.
{"latency":8,"location":"Bilbao","velocity":29.71822774471857,"timesta
mp":1496478583806}
{"latency":4,"location":"Donosti","velocity":8.76332237253636,"timesta
mp":1496478586084}
{"latency":4,"location":"Bilbao","velocity":41.99860113577196,"timesta
mp":1496478588807}
Donosti: Llevamos 23 medidas mayores de 50, de entre: 82 evaluadas.
{"latency":4,"location":"Bilbao","velocity":25.676145120206684,"timest
amp":1496478593807}
{"latency":5,"location":"Donosti","velocity":6.735454639058656,"timest
amp":1496478596085}
Bilbao: Llevamos 38 medidas mayores de 50, de entre: 84 evaluadas.
{"latency":12,"location":"Donosti","velocity":43.70408991530516,"times
tamp":1496478601085}
Bilbao: Llevamos 39 medidas mayores de 50, de entre: 85 evaluadas.
Donosti: Llevamos 24 medidas mayores de 50, de entre: 85 evaluadas.
{"latency":5,"location":"Bilbao","velocity":28.081836530262123,"timest
amp":1496478608808}
{"latency":4,"location":"Donosti","velocity":15.973606433668444,"times
tamp":1496478611086}
{"latency":4,"location":"Bilbao","velocity":19.56403077500906,"timesta
mp":1496478613808}
{"latency":4,"location":"Donosti","velocity":49.03281774884797,"timest
amp":1496478616086}
Bilbao: Llevamos 40 medidas mayores de 50, de entre: 88 evaluadas.
{"latency":12,"location":"Donosti","velocity":35.29172035636388,"times
tamp":1496478621087}
{"latency":4,"location":"Bilbao","velocity":15.177028619096848,"timest
amp":1496478623809}
{"latency":4,"location":"Donosti","velocity":15.111714325490553,"times
tamp":1496478626087}
{"latency":5,"location":"Bilbao","velocity":4.36763267380023,"timestam
p":1496478628809}
{"latency":5,"location":"Donosti","velocity":14.333553210822156,"times
tamp":1496478631087}
{"latency":6,"location":"Bilbao","velocity":21.833178714331698,"timest
amp":1496478633809}
{"latency":8,"location":"Donosti","velocity":20.40600482058542,"timest
amp":1496478636087}
{"latency":5,"location":"Bilbao","velocity":15.126161149572237,"timest
amp":1496478638810}

```
Donosti: Llevamos 25 medidas mayores de 50, de entre: 92 evaluadas.
{"latency":13,"location":"Bilbao","velocity":17.974725807812533,"times
tamp":1496478643810}
{"latency":8,"location":"Donosti","velocity":10.299903692506351,"times
tamp":1496478646088}
{"latency":7,"location":"Bilbao","velocity":46.166737694780274,"timest
amp":1496478648810}
{"latency":13,"location":"Donosti","velocity":34.550165212912425,"time
stamp":1496478651088}
Bilbao: Llevamos 41 medidas mayores de 50, de entre: 95 evaluadas.
{"latency":5,"location":"Donosti","velocity":26.33007036953284,"timest
amp":1496478656088}
Bilbao: Llevamos 42 medidas mayores de 50, de entre: 96 evaluadas.
Donosti: Llevamos 26 medidas mayores de 50, de entre: 96 evaluadas.
Bilbao: Llevamos 43 medidas mayores de 50, de entre: 97 evaluadas.
Donosti: Llevamos 27 medidas mayores de 50, de entre: 97 evaluadas.
{"latency":5,"location":"Bilbao","velocity":18.538361515944565,"timest
amp":1496478668812}
Donosti: Llevamos 28 medidas mayores de 50, de entre: 98 evaluadas.
{"latency":4,"location":"Bilbao","velocity":36.45517175802192,"timesta
mp":1496478673812}
{"latency":9,"location":"Donosti","velocity":39.64487381240855,"timest
amp":1496478676089}
{"latency":8,"location":"Bilbao","velocity":27.50728806633404,"timesta
mp":1496478678812}
{"latency":7,"location":"Donosti","velocity":49.454386399618514,"times
tamp":1496478681090}
```