Tesis Fin de Máster

EVALUACIÓN DE TÉCNICAS Y SISTEMAS DE PROCESAMIENTO DE DATA STREAMS

AUTOR: Santiago Valer Palacios

DIRECTOR: Sergio Ilarri Artigas

ÁREA: Lenguajes y Sistemas Informáticos

DEPARTAMENTO: Informática e Ingeniería de Sistemas

15 de Junio de 2011

Máster en Ingeniería de Sistemas e Informática

Curso académico 2010-2011

Escuela de Ingeniería y Arquitectura de Zaragoza

Universidad Zaragoza

EVALUACIÓN DE TÉCNICAS Y SISTEMAS DE PROCESAMIENTO DE DATA STREAMS

RESUMEN

Los Data Stream Management Systems (DSMS) o Sistemas Gestores de Flujo de Datos son una alternativa a la forma tradicional de recibir y tratar los streams de datos. Como se verá a lo largo del trabajo tienen muchos puntos en común con los Database Management Systems (DBMS) o Sistemas Gestores de Bases de Datos, entre otras cosas se sirven de lenguajes declarativos para gestionar los datos.

En este trabajo se pretende realizar una visión global, que permita extraer conclusiones sobre el estado actual de las soluciones planteadas en el tiempo y así servir de guía y ayuda en la elección del sistema más adecuado según el trabajo que un investigador quiera llevar acabo.

En primer lugar se realiza una descripción de los DSMS en general y sus características principales. A continuación se muestra información sobre los DBMS y una comparativa de estos con los DSMS.

Los sensores son los principales orígenes de data streams y la mayoría de los estudios y sistemas presentados cuentan con soluciones para uno o varios tipos de entornos con sensores. Por ello se presentan estos y las redes de sensores, con sus características fundamentales y los dominios de aplicación sobre los que a día de hoy se están aplicando.

En el tercer bloque se realiza una panorámica general, tipos, arquitecturas, lenguajes de consulta… sobre los DBMS, dado que la relación entre estos y los DSMS es evidente. Se realiza también un detalle de los tres DBMS más representativos con código libre. Por último en este mismo apartado, como conclusión se realiza una comparativa entre los DBMS y los DSMS.

A continuación se expone una descripción y comparativa cualitativa de los sistemas presentados teniendo en cuenta distintas características detalladas y justificadas.

En el quinto bloque se muestran resultados experimentales obtenidos con los distintos tipos de sistemas y una comparación entre ellos dónde ha sido posible por sus propias características. Se extraen de este punto y del anterior unas conclusiones conjuntas sobre la naturaleza, características y dominios de aplicación de los DSMS estudiados.

Por último se describe las siguientes vías de investigación y trabajo futuro a realizar como continuación y complemento a la labor que se ha iniciado con este trabajo y que sin duda puede resultar muy interesante para posteriores investigaciones.

MEMORIA.

Índice

1. Introducción

Diferentes propuestas acerca de cómo gestionar las llegadas de Data Streams a un sistema concreto, han sido presentadas en los últimos años.

Cada una de ellas ha sido detallada, evaluada y aplicada en determinados entornos, presentando resultados experimentales, detallados pero en pocas ocasiones comparados con otros sistemas ya existentes.

Además, los dominios de aplicación de los DSMS son amplios, cómo se verá en cada una de las soluciones que se recogen aquí, pero a menudo estas soluciones están pensadas para resolver sólo una parte o para aplicarse sólo sobre cierto tipo de dominios muy concretos. Por supuesto hay algunos sistemas de propósito general que también se estudian en el trabajo, pero no son la norma.

Para llevar a cabo esta evaluación de los DSMS existentes se comienza el trabajo mostrando información sobre los mismos a nivel general, sus características y su problemática propia ocasionada por el tipo de trabajo que se realiza con ellos y a menudo relacionada con los datos recibidos o la forma en que se reciben. Se realiza una introducción de los DSMS presentados hasta la fecha, de los primeros presentados a mediados de los 90 hasta los últimos aparecidos en el último año y un detalle de los puntos a tener en cuenta para evaluar y diseñar este tipo de sistemas, entre los que destacan las diferentes técnicas de ventanas.

Los sensores son los principales orígenes de data streams y la mayoría de los estudios y sistemas presentados cuentan con soluciones para uno o varios tipos de entornos con sensores. Por ello se presentan estos y las redes de sensores, con sus características fundamentales y los dominios de aplicación sobre los que a día de hoy se están aplicando.

En el tercer bloque se realiza una panorámica general, tipos, arquitecturas, lenguajes de consulta… sobre los DBMS, dado que la relación entre estos y los DSMS es evidente. Se realiza también un detalle de los tres DBMS más representativos con código libre. Por último en este mismo apartado, como conclusión se realiza una comparativa entre los DBMS y los DSMS.

A continuación se expone una descripción y comparativa cualitativa de los sistemas presentados teniendo en cuenta distintas características detalladas y justificadas.

En el quinto bloque se muestran resultados experimentales obtenidos con los distintos tipos de sistemas y una comparación entre ellos dónde ha sido posible por sus propias características. Se extraen de este punto y del anterior unas conclusiones conjuntas sobre la naturaleza, características y dominios de aplicación de los DSMS estudiados.

Por último se describe las siguientes vías de investigación y trabajo futuro a realizar como continuación y complemento a la labor que se ha iniciado con este trabajo y que sin duda puede resultar muy interesante para posteriores investigaciones.

En todo el texto y debido a la amplia utilización en la literatura del campo se mantienen algunos nombres en inglés, sin traducir al castellano.

2. Data Stream Management Systems (DSMS)

Un Data Stream Management System es un sistema que está diseñado para ejecutar consultas continuas sobre un flujo continuo de datos o *data stream*. Estos datos, que llegan en cada momento, permanecen sólo por un corto periodo de tiempo en memoria, pero se obtiene constantemente nuevos datos devueltos por las continuous queries. Los DSMS usan técnicas de ventanas o *windows* para manejar ciertas operaciones como puede ser la agregación.

Una ventana o *window* puede ser definida en términos físicos como un intervalo de tiempo, por ejemplo el último día, o en términos lógicos como un número de tuplas, por ejemplo los últimos 100 elementos.

Muchos prototipos de DSMS se han desarrollado en los últimos tiempos, algunos están especializados en un campo particular como la monitorización de sensores, aplicaciones web… y otros están orientados a propósitos generales.

El uso de estos sistemas para gestionar data streams es equivalente al uso de Sistemas Gestores de Bases de Datos (SGBD o DBMS) para gestionar bases de datos (BD).

2.1. Características de los DSMS

Existen una serie de características asociadas a los data streams, además de la llegada continua de datos ya mencionada.

Estas están relacionadas con su recepción, por ejemplo, estos datos pueden no llegar ordenados, su llegada en ocasiones puede ser muy rápida o a ráfagas, es decir, que esa recepción se produzca de una forma cambiante o no estacionaria en el tiempo.

El volumen de datos manejado puede ser muy grande y en muchas ocasiones puede resultar imposible realizar más de una lectura.

Sin embargo suelen tener una característica común, se necesita una respuesta inmediata una vez los datos se han recibido y procesado.

2.2. Problemática con la gestión de Data Streams

En los modelos de data streams, algunos o la mayoría de los datos de entrada no suelen estar disponibles para un acceso como el que se realiza a disco o memoria, sino como una llegada de flujos de datos o data streams continua. Estos problemas, según se mostró en [1] son:

- Los data streams se reciben en tiempo real.
- El sistema no tiene control sobre el orden de llegada de los datos que deben procesarse.
- Los data streams tienen un tamaño, potencialmente, infinito.
- Cuando un elemento de un data stream se ha procesado, este suele ser descartado y no puede ser recuperado fácilmente, a no ser que se almacene

explícitamente en memoria la cual es normalmente muy pequeña en comparación con el tamaño de un data stream.

Lógicamente, con una llegada continua de datos, la forma de realizar la gestión y explotación de estos se realiza mediante consultas continuas o *continuous queries*. Estas características únicas hacen que los requerimientos para gestionarse sean muy concretos como ya se expuso en [2]:

El modelo de datos y la semántica de las consultas deben permitir operaciones basadas en tiempo y orden.

La incapacidad de almacenar streams completos sugiere un uso de estructuras resumidas, por lo que los datos extraídos de las consultas no serán respuestas exactas.

Las consultas sobre streams no deben emplear operadores bloqueantes que intenten recibir todos los datos de entrada antes de producir resultados.

Debido al funcionamiento y las restricciones de almacenamiento, no es factible recuperar data streams. Esto es debido a que los algoritmos de ejecución sobre streams en tiempo real se restringen a pasar una sola vez sobre cada dato.

Las aplicaciones que monitorizan streams en tiempo real deben reaccionar rápidamente a datos con valores inusuales.

Las consultas, cuya ejecución se demore en el tiempo, pueden encontrarse con que aparecen variaciones en el rendimiento del sistema durante el su periodo de ejecución.

Permitir escalabilidad es un requisito obvio, por ello es indispensable compartir la ejecución de varias continuous queries.

2.3. Data Stream Management System estudiados

Desde que en el año 2000 apareció NiagaraCQ [3] [4], muchos son los sistemas publicados hasta la fecha con propuestas que gestionen el procesamiento de data streams.

Algunos de ellos como State Slace [5], Aurora [6], Gigascope [7], Nile [8] [9], PLACE [10], SPC [11] o STREAM [12] se encuentran abandonados o simplemente se dieron por concluidos y no se facilita soporte sobre ellos.

Sin embargo existen otros proyectos aún en funcionamiento hoy día o de los que, al menos, aún se facilita soporte como por ejemplo el ya mencionado NiagaraCQ [3] [4], Borealis [13] [14] [15] [16], Cougar [17] aunque está más orientado al procesamiento de consultas que a data streams propiamente dichos, Query Mesh [18] [19] [20] , SASE [21] [22] [23] [24] [25], TelegraphCQ [26] y TinyDB [27] [28] .

Por su relevancia desde su aparición o bien por su reciente propuesta se plantea en este trabajo un estudio más exhaustivo de Gigascope, Stream, NiagaraCQ, Borealis, TelegraphCQ, TinyDB, SASE.

Sistema	Entidad investigadora	Año
NiagaraCQ	University of Wisconsin-Madison	2000
TinyDB	Massachusetts Institute of Technology	2002
Cougar	Cornell University	2003
TelegraphCQ	University of California	2003
Aurora	Brandeis University, Brown University, and MIT	2003
Gigascope	AT&T Labs	2003
Nile	Purdue University	2004
STREAM	Standford University	2004
Borealis	Brandeis University, Brown University, and MIT	2005
SASE	University of Massachusetts Amherst	2006
SPC	IBM Research	2006
PLACE	Purdue University	2007
State-Slice	Worcester Polytechnic Institute	2008
Query Mesh	Purdue University	2008

2.4. Evaluar un DSMS

Como ya definieron Stonebraker, Cetintemel y Zdonik en 2005 [29], para poder evaluar o diseñar un sistema de procesamiento de data streams es necesario tener en cuenta una serie de requerimientos, en resumen estos ocho:

1. Procesar mensajes en streams, sin ningún requerimiento de almacenarlos para realizar cualquier operación.
2. Soportar lenguajes SQL para realizar consultas sobre streams de alto nivel.
3. Recuperación frente a streams imperfectas incluyendo datos perdidos o desordenados.
4. Un motor de procesamiento de streams que garantice resultados predecibles y repetitivos.
5. Capacidad de almacenamiento eficiente, acceso y modificación del estado de la información y combinarlo con data streams actualizados.
6. Asegurar que las aplicaciones están levantadas y disponibles, así como la integridad de los datos almacenados en cualquier momento a pesar de fallos.
7. Tener capacidad de distribuir procesos a través de múltiples procesadores y máquinas que permitan escalabilidad incremental.
8. Tener una gran optimización y un mínimo coste en el motor de ejecución para enviar respuestas en tiempo real a un gran volumen de aplicaciones.

	DBMS	Rule engine	SPE
Keep the data moving	No	Yes	Yes
SQL on streams	No	No	Yes
Handle stream imperfections	Difficult	Possible	Possible
Predictable outcome	Difficult	Possible	Possible
High availability	Possible	Possible	Possible
Stored and streamed data	No	No	Yes
Distribution and scalability	Possible	Possible	Possible
Instantaneous response	Possible	Possible	Possible

Figura 2.1 Requerimientos para evaluar un DSMS. Extraído de [29]

2.5. Ventanas en DSMS

Una ventana se puede definir, en el contexto de los DSMS, como una división del stream según un determinado límite, dentro del cual una consulta puede ser reformulada, por ejemplo, saber los mensajes que no han recibido respuesta 1 minuto después de enviarse.

En general se distinguen tres tipos de ventana o *windows* en la literatura de los DSMS como se ve en [30], *sliding*, *jumping* y *tumbling*. No obstante ya en la propuesta de TelegraphCQ [26] se discutió la existencia de las ventanas *landmark*. Se muestran en la figura 2.2 los esquemas de los distintos tipos de ventana.

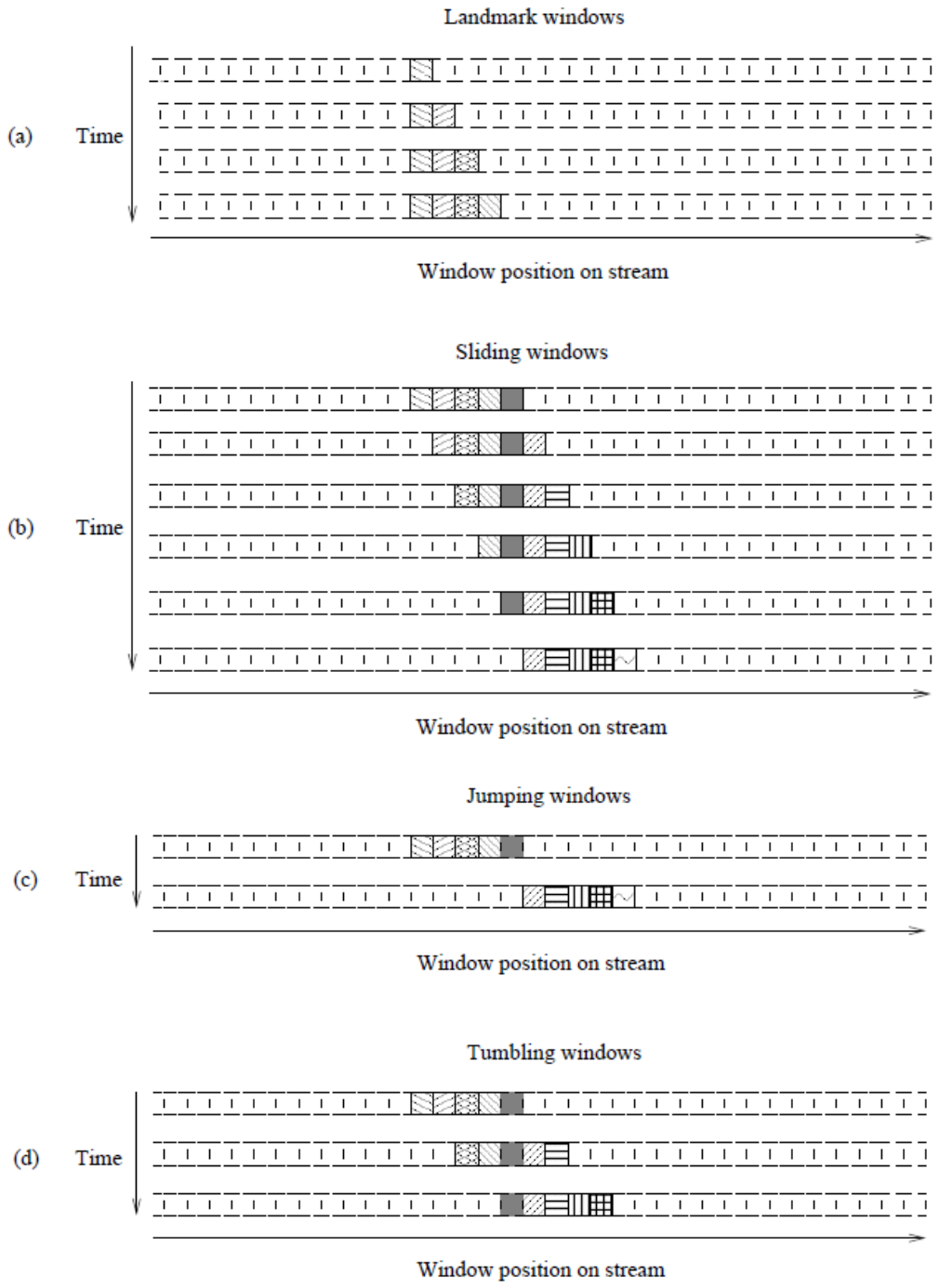

Figura 2.2. Tipos de ventana en DSMS. Extraída de [31].

2.5.1. Landmark Window

Tal y como se define en [32] una landmark window tiene un punto desde donde se mueve la ventana. Esta aumenta su tamaño mientras se añaden tuplas. No necesitan acceder a los datos históricos, que es una cualidad común para todas las técnicas de ventanas, para de esta manera evitar bloqueos, originados por ejemplo por operadores del tipo join. No se

necesita eliminar ninguna tupla que es lo que hace aumentar su número en la ventana, es decir, el tamaño de esta.

2.5.2. Sliding Window

La técnica de sliding window muestra que para una ventana de un determinado tiempo, todos los mensajes no respondidos serán borrados una vez transcurrido ese periodo. Esto quiere decir que una tupla dentro de los límites de una ventana permanecerá en ella un determinado tiempo antes de ser sobrescrita.

Ya se ha explicado en este trabajo anteriormente que una ventana puede ser definida en términos físicos como un intervalo de tiempo por ejemplo el último día, o en términos lógicos como un número de tuplas por ejemplo los últimos 100 elementos.

Las primeras necesitan asignar memoria dinámicamente dado que el comportamiento de los streams depende de fluctuaciones en la velocidad de la llegada de los datos.

Las ventanas lógicas por su parte, no usan restricciones de tiempo, sino un número de tuplas para determinar el tamaño de la ventana, es decir, el tamaño de memoria requerido se conoce a priori, y se puede reservar.

2.5.3. Jumping Window

En aquellos casos donde no se requiera recalcular como en las sliding windows, las *jumping windows* ofrecen una solución alternativa, rellenando la ventana con tuplas y ejecutando los cálculos. Después de haber calculado, se empieza a rellenar una nueva ventana. Así se elimina el re-cálculo de las tuplas, pero no asegura resultados correctos, por ejemplo, un mensaje puede tener su respuesta en la siguiente ventana.

2.5.4. Tumbling Window

Este tipo de ventanas se puede recalcular en un intervalo concreto de tiempo. Por ejemplo que sea mayor que la marca de tiempo de una sliding window, pero menor que la enésima marca de tiempo en una jumping window.

2.6. Sensores y redes de sensores

Uno de los principales dominios de aplicación de los DSMS son las redes de sensores, que son un conjunto de sensores que recogen datos y los transmiten de unos a otros hasta llegar a un receptor dónde serán procesados.

2.6.1. Sensores

Un nodo o dispositivo sensor, también llamado simplemente sensor, es un ordenador inalámbrico alimentado con baterías. Están diseñados para encargarse de medir y convertir una señal física, por ejemplo, un sonido, temperatura, luz… en una señal eléctrica que pueda ser manipulada, medida, amplificada o transmitida. Esta última acción es la que resulta más determinante en este trabajo.

En líneas generales se asume que la relación entre la entrada y la salida en un sensor debería ser proporcional, pero no suele ser totalmente lineal. Todos los sensores tienen un rango limitado de validez relacionado generalmente con las alternaciones del entorno que los rodea y que provocan retardos en las respuestas que emite.

Normalmente son pequeños y con un consumo energético muy bajo, dado que deben estar funcionando con baterías, en algunos casos, durante meses o años. Por ejemplo en TinyDB, por defecto, se pueden enviar 20 mensajes de 50 bytes por segundo.

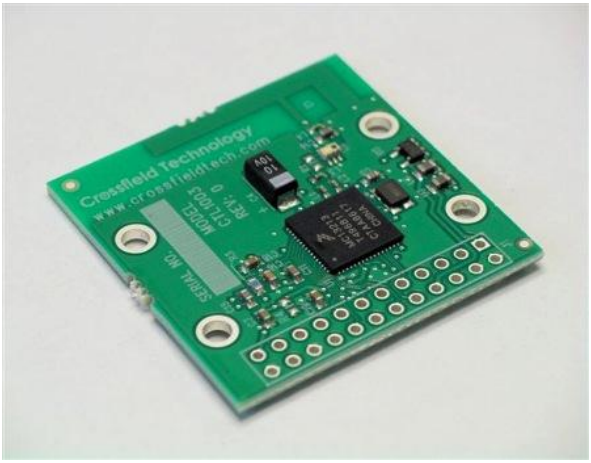

Figura 2.3 EH-Link, de MicroStrain, sensor inalámbrico con tecnología energy harvesting

2.6.2. Características de los sensores

Los sensores, además de su tamaño y de que estén alimentados por baterías, tienen otras características importantes que se pueden agrupar en estáticas (cómo actúa un sensor normalmente o con cambios lentos de lo que se quiere medir) y dinámicas (cómo actúa el sensor transitoriamente).

En el primer grupo se pueden englobar:

- Rango de medida. Conjunto de valores entre dos dados, máximo y mínimo, que puede captar el sensor con un cierto margen tolerable y conocido de error.
- Resolución. Es la capacidad de tiene es sensor de detectar cambios, variación de señal, en la variable que está midiendo.
- Linealidad. Relacionado con el concepto mostrado en la definición de sensor, se consideraría lineal a aquel sensor que tenga una constante de proporcionalidad entre las modificaciones entre los valores de entrada y los de salida.
- Repetitividad. Es la variación existente entre los valores de salida que facilita el sensor tras medir la misma entrada en mismas condiciones ambientales.
- Precisión. Es el margen existente entre la salida teórica que debería facilitar el sensor y la salida real.

- Sensibilidad. Es la propiedad que identifica la variación entre la señal de salida y la de entrada, siendo más sensible cuanto más varía la señal de salida ante pequeñas variaciones de la entrada.
- Ruido. Es en general cualquier tipo de perturbación en el entorno o el propio sistema de medida que afecte a la señal a medir.

Por otro lado estarían las características dinámicas:

- Velocidad de respuesta. Sería el retraso que tiene el sensor para emitir la salida ante una variación en la señal de entrada.
- Respuesta en variación de frecuencia. Es la capacidad del sensor para detectar las variaciones de las entradas cuando la velocidad en la detección de estas aumenta.
- Estabilidad. Es la desviación entre la salida teórica y la real, pero no debe confundirse con la precisión, dado que la estabilidad se medirá al variar los parámetros ambientales diferentes a los que se desea medir.

2.6.3. Redes de sensores: cableadas e inalámbricas

Ha pasado mucho tiempo desde las redes tradicionales de sensores cableadas. Emitían señales basadas en parámetros del entorno, por ejemplo en edificios, laboratorios y equipamientos. Incluso podían llegar a estar conectador a una red de área local o LAN que suministrase permanentemente energía eléctrica.

En este aspecto la ventaja de las redes inalámbricas de sensores es considerable. Permite desplegar y mantener miles de nodos si la necesidad de utilizar miles de cables y conexiones físicas [33].

Sin embargo tienen limitación de recursos energéticos, que fuerza un uso optimizado del procesamiento y las comunicaciones. Un hecho básico es contar con que el consumo de computación es en la mayoría de los casos muy inferior al consumo de las comunicaciones, por lo que se deberá siempre optimizar los recursos disponibles a la computación al máximo, para evitar así, hacer uso de las comunicaciones innecesariamente.

Entre otras características, se pueden destacar la limitación en la velocidad de procesamiento (MHz), limitación de capacidad de almacenamiento (Kbps), limitación del ancho de banda, operaciones desatendidas en los nodos, uso de comunicaciones broadcast en lugar de las usadas anteriormente de punto a punto, pérdida de paquetes por posibles pérdidas energéticas, nodos desplegados en grandes números aumentando las colisiones y cambios dinámicos de la posición de los sensores como los existentes, por ejemplo, en las redes vehiculares.

2.6.4. Dominios de aplicación

Con nuevas características, aparecen nuevos dominios de aplicación. Algunos de ellos pueden ser las aplicaciones militares, aplicaciones ambientales, aplicaciones de hábitats naturales, de negocios e industria, sanitarias, aplicaciones inteligentes en oficinas, hogares...

Hay toda una línea de sensores destinadas a las redes vehiculares y toma de datos de elementos en movimiento como peatones con dispositivos GPS, cuyas soluciones en ocasiones difieren del resto en la misma medida que su propia problemática.

3. Database Management Systems (DBMS)

Un sistema gestor de bases de datos, cuyas siglas en inglés son DBMS, es un software cuyo principal propósito es controlar la estructura, el almacenamiento, la integridad y la seguridad de los datos que almacena. Pueden trabajar con lenguajes de programación conocidos, como por ejemplo C, o incluir un lenguaje propio con el que programar.

3.1. Clasificación de DBMS

Existen varios tipos de clasificaciones para los DBMS, según diversos conceptos: modelo lógico, número de usuarios, de sitios y ámbito de aplicación.

Según el modelo lógico en el que están basados se puede hablar de:

Modelo de red. En él los datos se representan con conjuntos de registros y sus relaciones entre datos mediante enlaces, de forma similar a cómo funcionan los punteros. Los registros se organizan como grafos dirigidos, es decir un conjunto de vértices (registros), unidos por aristas (relaciones).

Modelo jerárquico. Parecido al modelo en red, dado que tanto los datos como sus relaciones están representadas por registros y enlaces. Sin embargo los registros no se organizan como grafos dirigidos si no como una colección de árboles.

Modelo relacional. Los datos se describen como relaciones representadas por tablas formadas por filas (tuplas que representan una ocurrencia) y columnas (atributos que representan las propiedades de las filas). Toda fila debe identificarse por un campo clave unívocamente.

No toda definición de tablas es válida, para ello deben emplearse diseños normalizados que garanticen que no aparezcan anomalías en el tratamiento de la base de datos. Sin embargo, en pro del rendimiento y otras consideraciones, algunos diseños pueden no estar completamente normalizados. Debe encontrarse un equilibrio entre normalización y rendimiento a la hora de diseñar una base de datos.

A día de hoy es el modelo más impuesto en el mercado, debido a su sencillez de manejo. En especial el uso del lenguaje SQL como estándar para gestionar los datos en este modelo ha supuesto un punto muy importante en su expansión.

Modelo orientado a objetos. No solo aplicado a los DBMS sino en general al desarrollo de software y de sistemas de la información. Se basa en un conjunto de objetos, los cuales almacenan valores en variables de ejemplares. También pueden contener código o métodos que operan en él. Si los objetos contienen los mismos tipos de valores y métodos están agrupados en clases, que son una definición de tipo para los objetos. En los lenguajes de programación existen los tipos abstractos de datos que serían una combinación de datos y métodos que constituyen una definición de tipos, como ocurre con los objetos.

En cuanto a las ventajas de los modelos orientados a objetos respecto a los relacionales se cuentan las siguientes:

- Tipos de datos complejos se manejan más fácilmente.
- El encapsulamiento permite facilitar su mantenimiento.
- El acceso a los datos es más fácil.

Las otras clasificaciones mencionadas se resumen a continuación:

Según el número de usuarios pueden ser: monousuario o multiusuario.

Según el número de sitios pueden ser centralizados o distribuidos (homogéneos o heterogéneos)

Según su ámbito de aplicación pueden ser de propósito general o específico.

3.2. Lenguajes en un DBMS

Existen cuatro tipos de lenguajes en los DBMS que permiten trabajar sobre una base de datos relacional:

Lenguaje de definición de datos o *Data Definition Language* (DDL). Es utilizado por el DBMS para identificar las descripciones de los elementos que componen la base de datos y especificar el esquema relacional interno.

Lenguaje de definición de almacenamiento o *Store Definition Language* (SDL). Especifica el esquema interno correspondiente a la base de datos almacenada.

Lenguaje de definición de vistas o *View Definition Language* (VDL). Se utiliza en el DBMS para la especificación de las vistas del usuario así como su correspondencia con el esquema conceptual.

Lenguaje de manipulación de datos o *Data Manipulation Language* (DML), considerado una combinación de los anteriores en las bases de datos relacionales, permite manipular operaciones de inserción, borrado y modificación de datos.

Un ejemplo de DML de alto nivel sería SQL que puede definir esquemas conceptuales, manipular los datos, definir las vistas del usuario y definir el almacenamiento.

3.2.1. Lenguajes de consulta

Ciertos sistemas estudiados emplean SQL como lenguaje de consulta, o añadiendo a este ciertas funcionalidades. Algunas de estas ampliaciones resultan, para los investigadores, suficientemente importantes como para dar a sus lenguajes un nombre propio, tal es el caso de TinySQL.

Otros sistemas comenzaron usando un lenguaje propio pero con el tiempo pasaron a centrarse en el propio sistema dejando de lado el desarrollo del lenguaje. Esto ocurrió por ejemplo con el DBMS conocido como PostgreSQL y que se verá en este trabajo más adelante. Este sistema comenzó con un intérprete del lenguaje QUEL que se basaba en Ingres, pero en

1995 se añadió el soporte para el lenguaje SQL a POSTGRES pasándose a llamar Postgres95 y que en 1996 tomaría PostgreSQL como su nombre definitivo.

3.2.2. TinySQL

Se trata en un apartado independiente el lenguaje TinySQL por la importancia que tiene en sí mismo como lenguaje empleado a nivel general y en particular en el DSMS TinyDB.

Es un lenguaje declarativo, dónde es el usuario el que indica lo que quiere pero no cómo lo quiere, como SQL para consultas específicas que serán ejecutadas en una red de sensores ejecutando TinyOS.

Los lenguajes declarativos tienen dos ventajas sobre los procedurales, donde el usuario sí que indica cómo quiere conseguir los datos. En primer lugar es sencillo aprender a leer y escribir consultas en este tipo de lenguajes. En segundo lugar es que con lenguaje declarativo el sistema subyacente puede cambiar la manera de ejecutar las consultas sin necesidad de cambiarlo en el propio lenguaje de consulta.

Esta segunda ventaja es especialmente importante en redes de sensores donde la implementación suele necesitar cambiar frecuentemente cuando un sensor se mueve, deja o se une a la red.

Sin embargo tiene ciertas limitaciones que se detallan a continuación:

- En la última versión las cláusulas WHERE Y HAVING pueden contener únicamente conjunciones simples sobre operadores de comparación aritméticos. Es decir, no están soportados los operadores booleanos OR ni NOT, o comparadores de cadenas similares a los existentes en SQL LIKE y SIMILAR.
- No se permiten sub-consultas.
- No se permite la asignación de alias a los nombres de las tablas, como si lo hace SQL con AS.
- Las expresiones aritméticas están limitadas a la forma *columna operador (+,-,*,/) constante*.
- Las consultas anidadas sólo pueden ser usadas junto a eventos.
- La cláusula ON EVENT da la oportunidad de ejecutar una consulta cuando ocurre cierto evento, esto permite ahorrar mucha energía en los estados de latencia o espera de los sensores entre evento y evento como se muestra en la Figura 2.1.
- Sin embargo TinySQL tiene ciertas características específicas para sensores como la cláusula TRIGGER ACTION que, siendo opcional, debería ser ejecutada en cualquier momento en que se produzca un resultado en una consulta. El tiempo, en ms, entre ejecuciones está especificado en la consulta gracias a la cláusula EPOCH DURATION. En caso de obviarse el tiempo por defecto son 1024 ms.

3.2.3. CQL

CQL es un lenguaje declarativo de consultas continuas basado en SQL implementado en la Universidad de Standford [34] e implementado en el prototipo de STREAM que permite

ejecutar la mayoría de sus funcionalidades. Los detalles de operadores se muestran comparándolos con GSQL en el siguiente punto.

3.2.4. GSQL

Es un lenguaje de consulta similar a SQL desarrollado para Gigascope, con el que gestionar aplicaciones de monitorización de red. Es un lenguaje exclusivo para streams pero se pueden crear y manipular las relaciones con funciones definidas por el usuario.

Los operadores primarios son selección, join, agregación y combinaciones. Todas están disponibles también en CQL de forma directa, excepto las combinaciones que son expresadas mediante el operador de unión. La agregación, será más complicada de expresar en CQL dado que requerirá de agrupaciones, proyecciones y *joins*. Este es un punto crítico que aunque en [34] se afirma que, sin ser una acción trivial, siempre se puede expresar sin dar más detalles.

Las restricciones en *joins* y agregaciones aseguran el sistema frente a bloqueos. En concreto el operador join debe contener un predicado con un atributo ordenado por cada stream afectado por el join, y el operador agregación debe tener al menos un atributo de agrupación por el que ordenar. En el caso de CQL, estos atributos de ordenación van intrínsecos en las marcas de tiempo.

3.2.5. Tipos de consultas

Las consultas soportadas por un lenguaje se pueden dividir en categorías, en concreto para TinySQL se pueden considerar:

Consultas de monitorización. Solicitan un valor para uno o más atributos continua y periódicamente.

Consultas de exploración. Consultas para examinar el estado de un nodo o un conjunto de nodos por ejemplo:

```
SELECT light
WHERE node_id = 1
ONCE
```

En este caso la cláusula *ONCE* evita producir un stream de datos, en lugar de eso se consigue una lectura simple del nodo cuyo id sea el 1.

Consultas anidadas. Son consultas dentro de consultas, como por ejemplo:

```
SELECT …
FROM …
WHERE attribute in
      (SELECT attribute FROM Table WHERE x < 10)
```

Este tipo de consultas no puede usarse en TinySQL directamente aunque hay otro tipo de estrategias para conseguir similares resultados.

Consultas de actuación. Se emplean cuando se quiere tomar una acción física concreta como respuesta a una consulta.

Envío offline. Utilizadas cuando se requiere conseguir alguna información de un suceso que ocurre más rápidamente de lo que los datos pueden ser transmitidos por radio.

3.3. Sistemas DBMS de código abierto.

En cuanto a las soluciones de DBMS existentes hoy día en el mercado se puede hacer una distinción entre aquellas propietarias y las de código abierto. Estas últimas serán en las que se centrará este punto del trabajo. Las soluciones abiertas son aquellas que a priori, pueden ser utilizadas según diversos licenciamientos, libremente aunque algunas planteen ciertas restricciones.

3.3.1. SQLITE

Desarrollado por Richard Hipp es un DBMS sin un proceso independiente como los DBMS basados en cliente-servidor. En su lugar se enlaza la biblioteca SQLITE con el programa desarrollado pasando a ser una parte del mismo. Se utilizan llamadas a subrutinas y funciones reduciendo la latencia en el acceso a la bases de datos, porque son más eficientes que la comunicación entre procesos. El punto negativo del sistema estaría en que el fichero único que almacena las definiciones, tablas, índices y datos, almacenado en el host principal, se bloquea al inicio de cada transacción, y que está limitado a 2Tb.

3.3.2. MySQL

Es un DBMS relacional, con un diseño multihilo para soportar eficientemente una gran carga. En principio la versión comercial solo aporta soporte técnico y la integración con otros software propietarios. A día de hoy es el DBMS más utilizado de las opciones existentes en software libre por su facilidad de uso y eficiencia. Precisamente el ser tan conocido es lo que permite que se hayan desarrollado muchas librerías y herramientas que permiten que MySQL se utilice con muchos lenguajes de programación.

En cuanto a sus características principales se destaca:

- El aprovechamiento real de los procesadores con varios núcleos (muy extendidos hoy día) y sistemas multiprocesador (muy comunes en la mayoría de servidores), gracias al multihilo mencionado anteriormente.
- Permite muchos tipos de datos para los campos.
- Hay aplicaciones en multitud de lenguajes como C, C++, Java, PHP…
- Permite una portabilidad cómodo y eficiente entre diferentes sistemas.
- Cada tabla puede definir hasta 32 campos clave.
- Alto nivel de seguridad de datos gracias a su gestión de usuarios y claves.

Sin embargo carece de ciertos aspectos que si tienen otros DBMS:

- No permite sub-consultas, aunque existen otras técnicas alternativas.

- Las claves ajenas definidas, no se gestionan de manera diferente a nivel interno a cómo se hace con el resto de campos.

3.3.3. PostgreSQL

Es un DBMS relacional - orientado a objetos cuyas primeras publicaciones datan de finales de los años 80 [35], [36], [37], aunque se comenzó en la Universidad de Berkeley de mano de Michael Stonebraker en 1982, con Ingres que evolucionaría en POSTGRES, proyecto terminado en 1994 tras la publicación de la cuarta versión. De estos comienzos es de dónde deriva PostgreSQL de 1997.

Las características principales del sistema son:

- Alta concurrencia.
- Gran variedad de tipos nativos.
- Implementación del estándar SQL92/SQL99.
- Estructura de datos *array*.
- Funciones de diversa índole incluso permite definir funciones propias y disparadores o *triggers*.
- No tiene objetos pero permite la herencia (característica principal de los sistemas orientados a objetos) entre tablas. Es por ello por lo que se considera un sistema relacional – orientado a objetos.
- Gestión de seguridad mediante la definición de usuarios y permisos independientes para cada uno.

Por otro lado carece de características imprescindibles en muchos casos, como la velocidad de respuesta, aunque es constante incluso en bases de datos de gran tamaño lo que pasaría a considerarse una ventaja en lugar en un hándicap.

3.3.4. Comparativa MySQL – PostgreSQL

Se eligen estos dos sistemas para esta comparativa por ser MySQL el más extendido en el mundo de los SGDB y PostgreSQL uno de los más utilizados en los DSMS planteados en el trabajo.

Resumiendo los puntos positivos de PostgreSQL se puede decir que:

- Tiene una gran escalabilidad, ajustándose al número de procesadores y cantidad de memoria para optimizar el sistema y soportando mayor cantidad de peticiones simultáneas correctamente, en algunos casos el triple, según ciertos benchmarks.
- Permite sub-consultas, transacciones y recuperación frente a desastres, siendo más eficaz.
- Comprueba la integridad referencial y permite almacenar procedimientos en la base de datos. Esta es una funcionalidad muchas veces restringida a DBMS de alto nivel como Oracle.

Pero por otro lado:

- Consume muchos recursos.
- Por defecto se limita a 8Kb por fila, aumentables a 32Kb penalizando el rendimiento.
- Puede entre el doble y el triple de lento que MySQL

Resumiendo los puntos positivos de MySQL se puede decir que:

- Es rápido en la ejecución de operaciones, mejorando sustancialmente el rendimiento.
- Baja consumo.
- Buenas utilidades de administración, fácil instalación y configuración.
- Bajas posibilidades de contener datos corrompidos.

Sin embargo tiene una serie de inconvenientes:

- No soporta transacciones, sub-consultas ni rollbacks
- No maneja integridad referencial con muchos accesos pues no tiene buena escalabilidad.

En conclusión se puede decir que, tras evaluar las ventajas e inconvenientes de estos sistemas, cada uno de ellos tiene propiedades que los hacen buenos en distintos campos de aplicación. Será este el punto fundamental a la hora de elegir uno u otro y equivocarse en la decisión haría descender la efectividad y la eficiencia del software y del propio DBMS.

3.4. Diferencias entre DSMS y DBMS.

Aunque son sistemas que por tradición se han comparado a lo largo de su existencia, existen entre ellos ciertas diferencias que se detallan a continuación.

En lugar de relaciones persistentes, los DSMS se centran en manejar streams pasajeros. Esto directamente implica en que el espacio en disco no es un problema, dado que los datos dejan el sistema, normalmente, a gran velocidad. Aunque al ser esta una postura opuesta a los DBMS, algunos DSMS han integrado un DBMS como una parte de ellos. Esto facilita la posibilidad de unión entre streams y tablas, entre otros. Un ejemplo que se verá en siguientes puntos es TelegraphCQ y PostgreSQL.

Mientras que los DBMS acceden a los datos una vez por consulta, los DSMS se sirven de continuous queries, que obtienen tuplas continuamente de los streams. Dicho de otro modo, los DBMS permiten consultas temporales sobre datos persistentes, los DSMS permiten consultas persistentes sobre datos temporales.

Desde que los datos llegan en forma de stream, los DSMS revisan las tuplas como una secuencia lineal sin tener acceso a los datos antes o después del intervalo de acceso. Los datos de los DBMS están almacenados en una base de datos, obviamente, y pueden se puede acceder a ellos aleatoriamente mediante la especificación de los bloques a leer. En ciertas operaciones como *joins* y *aggregations* esto significa que el DBMS estará bloqueado mientras se están ejecutando. No es posible sobre un stream lineal de datos. Operando sobre streams, los DSMS, pueden soportar *windowing* para bloquear los operadores, lo que significa que

aunque el stream sea infinito, los cálculos serán ejecutados sobre pequeñas partes de ese stream. Estas partes están localizadas en la memoria principal, lo que implica limitaciones a considerar en el tamaño de la ventana y la precisión.

Los DSMS no suelen estar orientados al almacenamiento de las tuplas que llegan al sistema, que está limitado por el tamaño de la memoria disponible. Esto por tanto, es un problema a considerar con las características de entrada y salida o I/O del disco. Como los streams suelen llegar a gran velocidad, no se debe contemplar un uso elevado del tiempo I/O del disco. Por tanto sólo se deben usan algoritmos de un paso o *one-pass algorithms.*

Para permitir cierta optimización, los DBMS tienen un conjunto de reglas que se introducen en el proceso de reescritura de las consultas. Este tipo de reglas también juegan un papel importante en los DSMS, pero necesitan adaptar sus streams de la misma manera, por ejemplo redistribuyendo los tamaños de las consultas.

Dado que los data streams pueden llegar en ráfagas muy rápidas, esto implica que ciertos datos no puedan ser computadas, por diversos factores como puede ser la ejecución de consultas demasiado complejas. Por lo tanto, esto también implica que los DSMS deban soportar algoritmos de aproximación. Los DBMS por su parte, no pueden garantizar que los resultados sean mostrados dentro de una fecha límite, por ejemplo ciertas consultas complejas usando varias relaciones sobre muchos gigabytes de datos pueden tardar varias horas en terminar en un DBMS.

4. Evaluación cualitativa de los DSMS estudiados

Hace varios años que los sistemas de data streams están en boga [2]. De los sistemas aún existentes hoy día se ofrece a continuación una panorámica general, siempre en función a la aplicación para la que están pensados, los valores de entrada que aceptan, los operadores usados, el técnica de ventanas empleada, y si es posible su capacidad de optimización y adaptación.

4.1. NiagaraCQ

Mientras Aurora y Gigascope se centran en cadenas de datos sencillas, NiagaraCQ [3] soporta data streams suministrados por páginas web, a las cuales se pueden lanzar *continuous queries*. Este DSMS permite agrupar un número muy elevado de consultas continuas agrupándolas según las cualidades que comparten. Por tanto será necesario encontrar estas similitudes. Esto se consigue gracias a un lenguaje que, a través de sintaxis e instrucciones XML, permite ejecutar múltiples consultas continuas.

Se define como un sistema escalable que permite consultas continuas sobre bases de datos de gran tamaño. Se afirma que el sistema permite escalar la propuesta presentada a consultas continuas ejecutadas a nivel de Internet, es decir, para que soporte millones de ellas, agrupándolas. Además se define como un sistema fácil de implementar y eficiente.

Para ser implementado basta con usar un motor de consulta general con mínimas modificaciones, donde sólo se cambia la porción de fichero XML que deba ser actualizado, ahorrando así recursos de computación. Está implementado con un subsistema del proyecto Niagara como se ve en la siguiente figura 4.1.

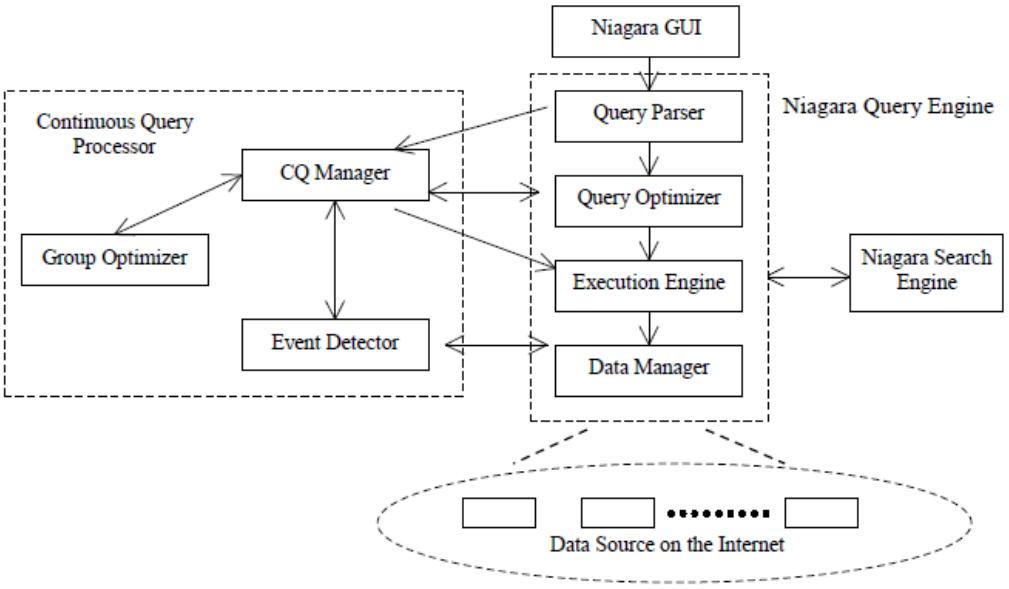

Figura 4.1. Arquitectura del sistema NiagaraCQ. Extraída de [3]

4.2. TinyDB

Es un procesador de consultas para redes de sensores, desarrollado por el MIT. En 2003 la Universidad de Berckley publicó [27] dónde se presentaban las principales aportaciones de la versión 1.1 del sistema, así como sus ventajas para el procesamiento de consultas en red. Entre ellas destacan las consultas sobre memoria flash, el soporte para sensores capaces de medir temperatura, luz… una nueva interfaz de red modular y agregación, gestión de consumo, sincronización de tiempo, consultas basadas en eventos, TASK (Tiny Application Sensor Kit), soporte para simulador y nuevas herramientas de compilación, testeo y estado.

Algunas de las características de TinyDB son la gestión de metadatos, consultas en alto nivel, topología de red, consultas múltiples y despliegue incremental mediante la compartición de consultas.

En cuanto a las desventajas del sistema se pueden identificar varias, entre ellas que no posee un modelo de programación general, que los modelos propuestos son difíciles de modificar y que hay una estrecha relación entre el trazado de rutas y las consultas. Además, para este trabajo en particular, es un sistema muy centrado en la evaluación de consultas, en concreto en redes de sensores, pero no orientado a data streams.

4.3. TelegraphCQ

Desarrollado en la Universidad de Berkeley, es un DSMS de propósito general. Construido como extensión del Data Base Management System o DBMS relacional PostgreSQL, soporta continuous queries sobre data streams. El formato de los data streams está definido, como cualquier tabla en PostgreSQL, en su Data Definition Language o DDL y se crea usando CREATE STREAM.

Las consultas sobre streams, permiten la sintaxis SELECT, incluyendo predicados adicionales como RANGE BY, SLIDE BY y START AT para especificar el tamaño de ventana. Cada stream tiene un atributo especial de tiempo que TelegraphCQ emplea como marca temporal de las tuplas en las operaciones con ventana.

No obstante TelegraphCQ tiene técnicas que difieren de PostgreSQL, es decir, que no está implementado con todas las funcionalidades de este último. Su lenguaje de consulta es StreaQuel, muy similar a SQL excepto por la semántica de ventanas. Su principal adaptabilidad esta creada para un módulo llamado *Eddy*, el cual envía y recibe filas que se procesan con distintos operadores. La técnica de ventanas permite usar deslizamiento, saltos y caídas de ventanas. Al contrario que STREAM, TelegraphCQ sólo gestiona un tipo de stream, pero provee funcionalidades de almacenamiento de streams en disco, pudiendo estos ser consultados con PostgreSQL posteriormente.

Principalmente está diseñado para:

- Planificación y gestión de recursos para grupos de consultas.
- Soporte para datos externos al núcleo.
- Sistemas con adaptabilidad variable.

- Soporte dinámico para QoS.
- Distribución y procesamiento basado en clúster paralelo.

Aunque TelegraphCQ no permite consultas anidadas, existe la alternativa de usar construcciones con la cláusula WITH. Tampoco se permiten *autojoins* ni *selfjoins* pero también existe una alternativa consistente en crear dos streams idénticos y hacer un join entre ellos.

4.4. Aurora

Implementado entre las universidades de Brown y Brandeis y principalmente orientado a las consultas sobre datos devueltos por sensores, aunque como datos de entrada se emplean cadenas y tablas. En concreto se refiere a tablas estáticas como término para describir ventanas con cadenas con tamaño ilimitado.

Tiene un álgebra de consultas llamado SQuAl (Stream Query Algebra) donde un operador manipula cajas, representadas como un conjunto de operadores, los cuales se usan entre otras cosas para filtrar, ordenar, mapear, agregar y unir.

Emplea diversas técnicas de optimización como insertar proyecciones, combinar y reordenar cajas. En tiempo de ejecución, un monitor QoS investiga los streams y averigua si necesitan optimización. Hay una versión de Aurora conocida como Aurora* cuando se usa en acoplamientos ajustados entre varias máquinas.

Tiene un motor de búsqueda para pocos nodos pero Medusa [38] ofrece una solución para distribuirlo a múltiples nodos y redes organizativas. Está desarrollado en el MIT y se centra en los cambios entre redes así como multiplexación entre varias conexiones.

4.5. Borealis

Hereda elementos de Aurora y Medusa, y además está desarrollado por las 3 entidades desarrolladoras de estos: MIT; Universidad de Brown y Universidad de Brandeis. Maneja tanto el procesamiento de consultas entre nodos como la distribución de varios nodos sobre grandes redes. Esto permite mayor escalabilidad, picos de gran carga, alta disponibilidad, monitorizar el estado del sistema y gestionar la tolerancia a fallos. Todo ello ayuda a Borealis a actuar dinámicamente.

4.5.1. Alta disponibilidad en Borealis

Tal como se ve en [14] existen una serie de modelos básicos de alta disponibilidad:

Recuperación precisa. Después de un fallo la salida debe ser la misma que la que habría habido sin fallo. Muy requerida en servicios financieros.

Recuperación rollback. La salida que se produce tras un fallo es equivalente a una ejecución sin fallo, aunque puede contener tuplas duplicadas, no se contempla la posibilidad de perder información pero los resultados pueden ser imprecisos. Este sistema suele ser requerido en sistemas de alarmas: incendios, anti intrusión…

Gap recovery. Es la forma más débil de recuperación donde se acepta pérdida de datos para mejorar la ejecución. Monitorización de entornos basados en sensores donde los últimos datos son los válidos, suelen ser los que hacen uso de este tipo de recuperaciones.

Según el comportamiento entre el servidor primario y el de seguridad se definen 4 aproximaciones en Borealis que garantizan la recuperación del sistema.

Amnesia. No contempla ningún preparativo para los fallos. Si se detecta una caída del servidor primario, el de backup comienza desde un estado inicial.

Passive Standby. Cada servidor primario crea puntos de control periódicos en los servidores de backup. Si el primario falla, el de backup inicia desde el último punto de control recibido.

Active Standby. El servidor de backup procesa en paralelo con el primario todas las tuplas, solo que las tuplas resultantes en el de backup no se envían como stream hasta que se detecta que el primario ha caído.

Upstream backup. Conserva las tuplas en la cola de salida mientras los nodos vecinos todavía las estén procesando. Si el servidor falla, un servidor de backup vacío reconstruye el último estado del primario que ha fallado a partir de los logs y de las tuplas de las colas de los servidores predecesores.

4.6. Gigascope

El Data Stream Management System (DSMS) conocido como Gigascope se desarrolló en AT&T para monitorizar el tráfico de red en los ISP's. El sistema se considera distribuido dado que algunos operadores de consulta se envían a los routers para recoger información que resulte de interés. Emplea un leguaje de consulta conocido como GSQL, el cual soporta selección, unión, agregación y fusión de cadenas.

Como alternativa al modelo basado en relaciones, opera sobre data streams directamente en lugar de transformar los datos. Además intenta permitir operadores de bloqueo asumiendo monotonía y un orden en los atributos de unión. Si por ejemplo una unión de dos cadenas R y S tiene una secuencia incremental de *a* números, se puede calcular dicho *join* simplemente mirando los valores de a tal y como llegan.

La intención de Gigascope es obtener datos de líneas de fibra óptica simples. Gracias a esto el operador de fusión se usa para conseguir unir los dos streams antes de hacer el *join*. Se considera una técnica de optimización además de reordenar los operadores los cuales al mismo tiempo también optimizan el sistema.

4.7. STREAM

Es un prototipo de DSMS de propósito general implementado en la Universidad de Stanford. Permite la aplicación de un gran número de consultas continuas y declarativas a datos estáticos, es decir, tablas o a datos dinámicos, es decir, streams.

Se emplea CQL (Continuous Query Language), un leguaje de consulta declarativo que deriva de SQL y que puede procesar continuous queries. Emplea tres tipos de ventanas deslizantes sobre streams: basadas en tiempo, basadas en tuplas o particionadas (similares al GROUP BY en SQL).

Gracias a las ventanas deslizantes desbloquea las cadenas de datos, con un conjunto de operadores: stream-relación, relación-relación, y relación-stream. Este último emplea tres definiciones diferentes de stream. ISTREAM muestra las filas insertadas en el stream, DSTREAM muestra las filas eliminadas del stream y RSTREAM muestra la relación dentro de la ventana. También soporta sub-consultas, aunque no se permitan en CQL, se expresan como vistas SQL.

Su principal meta es aprovechar los recursos compartidos y la adaptabilidad de aquellas consultas con sub-expresiones comunes. Los streams de entrada son filas y los tipos de datos pueden ser integer, char(), float o byte.

4.8. SASE

Diseñado y desarrollado por la Universidad de Massachusetts en colaboración con la de California. Es un sistema de procesamiento de eventos complejos sobre streams en tiempo real. Diseñado para especificar una aplicación lógica como la transformación, concebir nuevas técnicas para el procesamiento de consultas para implementar el lenguaje eficientemente y desarrollar un sistema comprensivo que recoja, limpie y procese datos RFID para almacenar los relevantes, la información temporal así como su almacenamiento para futuras consultas.

En la figura 4.2 se puede observar un esquema de la arquitectura de SASE, consistente en tres capas. La más baja para los dispositivos RFID, una siguiente que recibe los datos y genera los eventos y por último una tercera dónde la mayoría de los procesamientos de datos tiene lugar.

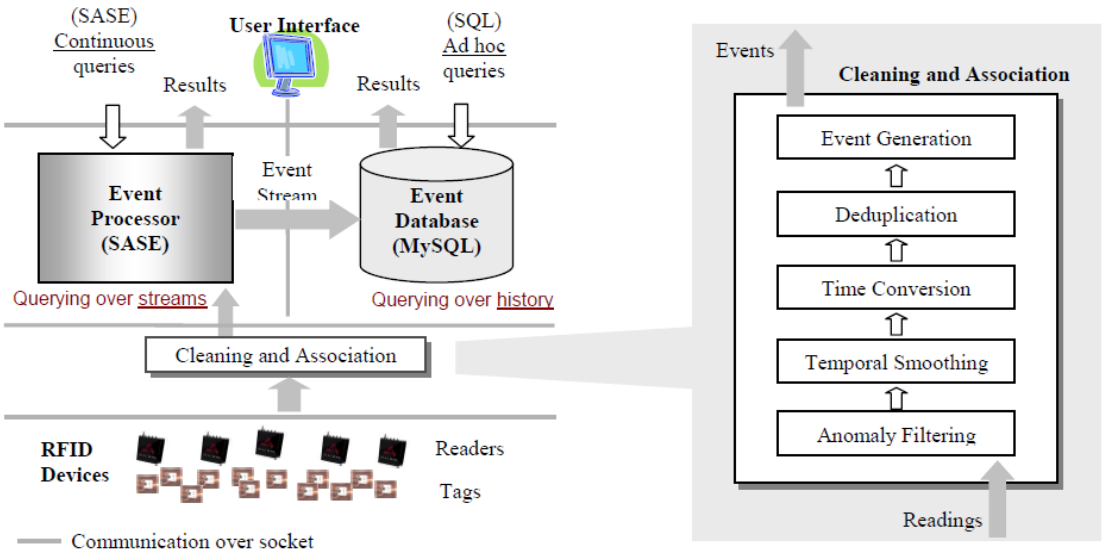

Figura 4.2. Arquitectura del sistema SASE. Extraído de [22].

En [24] se presenta adicionalmente SASE+, un lenguaje declarativo para especificar patrones de eventos complejos sobre streams, presentado como muy ágil respecto a otras propuestas del mercado según su escenario de demostración.

4.9. Query Mesh

Desarrollado en colaboración de la Universidad de Purdue y el Instituto Politécnico de Worcester en el 2008 y con publicaciones posteriores en 2009.

Dado que en la vida real los conjuntos de datos no son uniformes, seleccionar un plan de ejecución simple puede dar como resultado una consulta inefectiva para grandes secciones de los datos existentes.

En su solución pretenden calcular múltiples rutas (o planes de consulta), cada uno diseñado para un particular subconjunto de datos distinguiendo diferentes propiedades estáticas. Tiene inducido un modelo clasificador basado en la ejecución de esas rutas y las características de los datos. Estas son sus aportaciones, pero por distanciarse demasiado del resto de propuestas debe ser tenido en consideración en futuros trabajos de evaluación más centrados en las propuestas con procesamiento de consultas con multi-ruta adaptativa.

En este ámbito sería interesante abarcar además de Query Mesh a Eddies [39] y Content-based routing (CBR) [40]. Estos son las mejores aproximaciones hasta la fecha desarrolladas en CAPE [41].

4.10. Tabla comparativa

En las figuras 4.3 y 4.4 se muestra una tabla comparativa con las principales características documentadas y testeadas de los sistemas descritos en los anteriores puntos.

En ellas se muestran una serie de variables elegidas por ser las principales características de un DSMS. En primer lugar el tipo de consultas que se permiten realizar pues será determinante según el tipo de datos que se deseen extraer, especialmente consultas agregadas.

Si permiten o no detección de eventos pues es fundamental en muchos dominios de aplicación aunque no lo sea en otros, por ello se detalla para qué tipo de estos está orientado.

Si permite o se ha estudiado su aplicación a entornos de redes de sensores en alguna de sus variantes.

Por último con qué tipo de lenguajes de consulta trabajan y si tienen un interfaz gráfico de usuario disponible.

Sistema	Tipos de queries	Detección de Eventos	Contextos de aplicación	Definición de sensores SensorML	Consultas agregadas	Lenguajes de consulta	GUI
NiagaraCQ	Selección, Continuas, Agrupadas y Join	Event Detector (Data source changes y Timers events)	Búsquedas y BD en Internet. Grandes BD en general	NO	NO	Niagara XML-QL, SEQL	Java
TinyDB	Selección, Join, Project, Basadas en eventos y en tiempo de vida, Múltiples, Actuación, Estado, Exploración, Having, Group by, Relacionales y Alto Nivel. No subconsultas ni anidadas.	Iniciar la colección de datos. Parar las consultas.	Redes de sensores. Requerimiento de bajo consumo. Redes en general.	TinyML (SML sobre XML)	Datos agregados. Consultas agregadas (Aproximación distribuida)	TinyDB language. TinySQL.	Java
TelegraphCQ	No anidadas pero equivalente con WITH. No single Join pero si Join de 2 streams iguales. Selección y Group by. No subconsultas. Centrado en gestionar continuous queries sobr un gran volumen variable de data streams.	SI	Propósito general. Planificar gestión de recursos para grupos de consultas. Soporte a datos externos al núcleo. Adaptabilidad variable. Soporte dinámico QoS. Distribución y procesamiento basado en clientes paralelos.	NO	Si, con clausula *windowis*	StreaQuel	Monitorización online sin estado de recursos, p.ej. memoria usada en las consultas.
Borealis	Selección, Join, Agregadas y Continuas.	NO	Propósito general. Gestión de carga de procesamiento. Alta disponibilidad. Operaciones distribuidas. Redes de sensores.	NO	Si	No. SQuAL (Álgebra heredada de Aurora) Borealis Stream Processing Engine	C++ & XML

Figura 4.3. Tabla comparativa DSMS 1

Sistema	Tipos de queries	Detección de Eventos	Contextos de aplicación	Definición de sensores SensorML	Consultas agregadas	Lenguajes de consulta	GUI
Gigascope	Selección, Continuas, Ventana deslizante, Join, Group by y Agregación	NO	Aplicaciones en red. Detección de intrusión en red. Análisis de información de routers. Investigación de red. Monitorización. Redes de sensores.	No, pueden desarrollarse aplicaciones que lo hagan.	Si	GSQL SQL	Si
STREAM	Selección, Join, Continuas, Proyección, Filtros, Agregadas, Group by, Unión, Except, Named queries (Views). No ventan deslizante, ni having, ni subconsultas tras WHERE ni Having	SI	Propósito general. Análisis financiero. Monitorización de red. Producción. Redes de sensores. Streams rápidos. Carga variable. Recursos limitados.	NO	Si, pero no en todas las partes de la sentencia.	CQL para consultas complejas. SQL para sencillas.	Basada en Web. C++
SASE	Selección, Ventana, Negación, Transformación y Filtro.	Si. SASE+	Dispositivos sensores. Wireless motes. Lectores RFID. Retail Mangement. Healthcare.	NO	NO	SASE+ Lenguaje de eventos. SQL-TS Cayuga	Java
Query Mesh	Selección, Join y Filtros.	SI	Tráfico de datos en red. Datos en la bolsa. Redes de sensores. Objetos móviles en red. Transporte.	NO	NO	SQL	Java (CAPE*)

Figura 4.4. Tabla comparativa DSMS 2

5. Evaluación de rendimiento de los sistemas estudiados

En este punto se va a realizar una recopilación y evaluación de los resultados experimentales que se han obtenido de los sistemas propuestos hasta la fecha.

5.1. TinyDB.

Los desarrolladores de TinyDB implementaron a su vez Tiny Application Sensor Kit (TASK) con un conjunto de herramientas de configuración e interfaces de diseño para usuarios que hicieran más fácil la interacción con redes de sensores mientras se ejecutaba TinyDB. Esto incluye:

- Configuración y despliegue de herramientas de gestión, gracias a las cuales los usuarios pueden colocar sensores en un mapa y recoger varios tipos de metadatos sobre ellos.
- Herramienta de trabajo pensada para PDA's que permiten realizar estudios de campo y diagnósticos a los nodos sensores.
- Una interfaz simplificada que permite añadir consultas al sistema.
- Completa integración con un motor de base de datos relacional que recoja las consultas, los comandos y los resultados que se envíen o reciban de la red.
- Herramientas de extracción de datos diseñada para facilitar la recogida de esos datos mostrándolos en aplicaciones de usuarios finales.

TinyDB incluye soporte para el simulador de TinyOS, también conocido como TOSSIM. Este simulador dispone de una aplicación GUI en java que puede ser ejecutada con PostgreSQL en máquinas Linux o bien con Cygwin bajo Windows. En ambos casos, pero obviamente con sus particularidades, basta con descargar e instalar PostgreSQL, habilitar las conexiones remotas, instalar el driver JDBC, crear una base de datos y un usuario TinyDB y crear la tabla de consultas.

Una de las principales aportaciones es la adaptabilidad de la velocidad con la que se toman muestras en los sensores. Esto produce un ahorro energético de casi el 30% en un acelerómetro y de un 20% en un sensor magnético según [28].

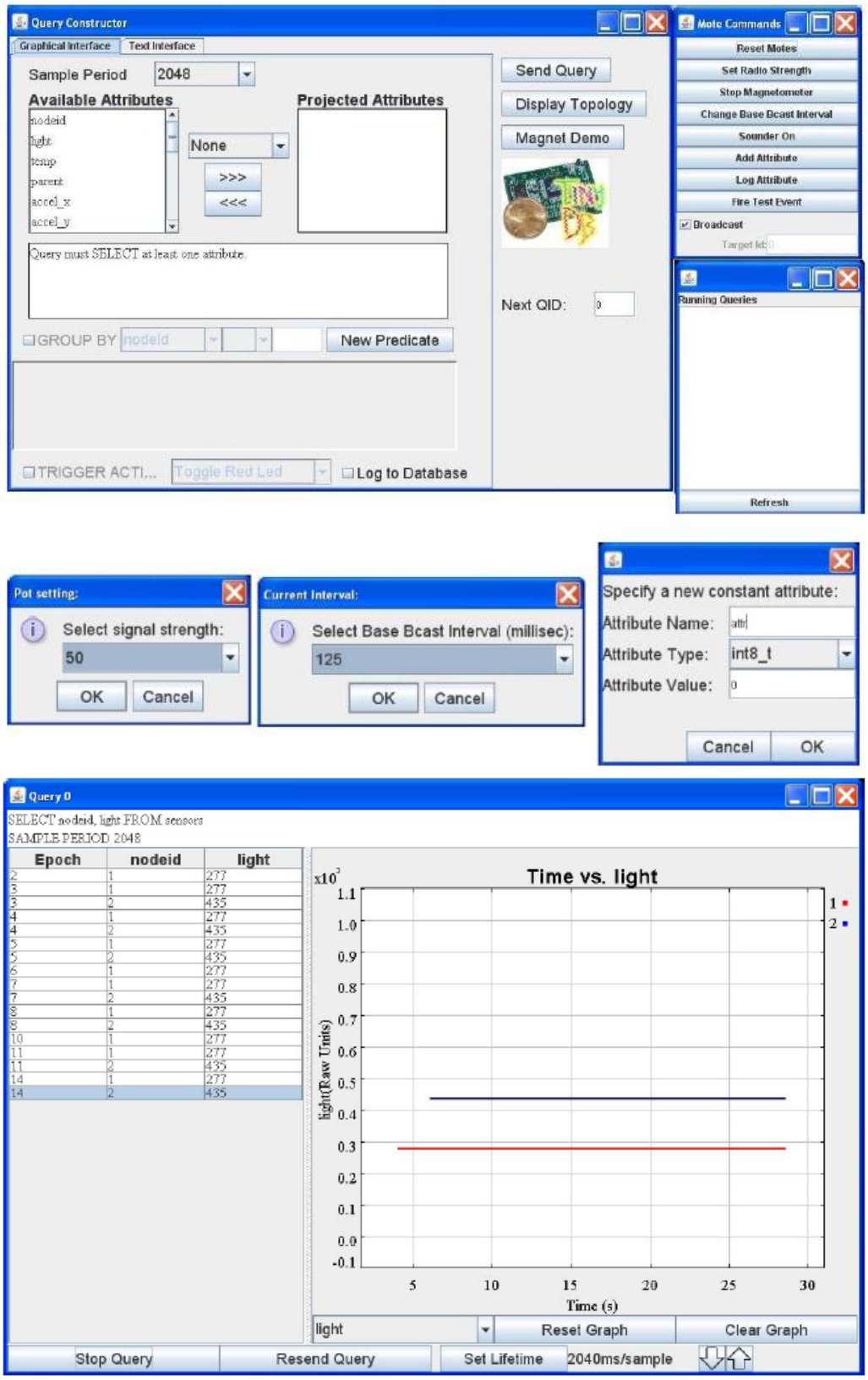

Figura 5.1. Pantallas del GUI de TinyDB. Construcción, configuración y resultados.

5.2. Sistemas de propósito general. STREAM y TelegraphCQ

Multitud de experimentos se han realizado entre los sistemas de DSMS existentes tanto de forma individual como comparativas entre ellos. Algunos como STREAM y TelegraphCQ de propósito general resulta complejo compararlos fuera de un ámbito concreto y específico de investigación. Tal fue el caso de [42] donde se utilizó estos DSMS para analizar datos sobre el consumo de energía eléctrica.

Para ello se instala STREAM y TelegraphCQ y realizan testeos durante 150 días, obteniendo medidas del consumo de 3 ciudades diferentes, aunque se considera escalable a un número mucho mayor. Se envían tuplas por cada ciudad cada 2 segundos, con un identificador, hora, fecha, índice mostrado por el medidor y más información adicional.

Se investiga con diferentes experimentos para hallar con ambos sistemas:

- Consumo de los últimos 5' (minuto a minuto) agrupado por medidor o por ciudad.
- Consumo histórico (minuto a minuto) agrupado por medidor o ciudad, comenzando en un punto fijado.
- Alarma horaria si se excediese el consumo normal, según la temperatura.

Una vez realizados los experimentos se extrae las siguientes conclusiones:

Por claridad y facilidad a la hora de escribir consultas, TelegraphCQ resulta mucho mejor que STREAM, gracias a la gestión de datos que realiza su sistema embebido PostgreSQL.

TelegraphCQ permite reutilizar resultados de consultas como unos streams, gracias a lo cual se puede exportar los resultados a un fichero que se almacene en un formato específico.

Además se pueden añadir dinámicamente consultas mientras otras están siendo ejecutadas, lo que no es posible en STREAM.

Por su parte STREAM tiene una interfaz gráfica (GUI) con aportaciones únicas como es el monitorizado del sistema, que permite a los administradores la posibilidad de inspeccionar el sistema en tiempo real, viendo cómo se gestionan las continuous queries.

Por tanto para este experimento concreto se puede concluir que TelegraphCQ es más útil que STREAM, dado que permite definir más fácilmente las consultas deseadas para llevas a cabo el experimento, siendo esta una ventaja común para la mayoría de sistemas, no solo aquellos relacionados con el consumo eléctrico.

Sin embargo aquellos sistemas en los que la monitorización y control de parámetros en tiempo real, como por ejemplo la cantidad de memoria que está siendo consumida por las consultas, sea un requisito indispensable deberán elegir STREAM.

No obstante ninguno de ellos tendría la funcionalidad expresa de los DSMS distribuidos como Borealis, para control de medidas automáticas, como los necesarios en redes de sensores distribuidos en la naturaleza.

5.3. TelegraphCQ como analizador de tráfico de red

A pesar de lo anterior, sí que se ha empleado TelegraphCQ con ciertas funciones de monitorización para evaluar los resultados devueltos por dispositivos sensores en tareas on line. Tal es el caso de [43] en el que se definieron experimentos para evaluar TelegraphCQ como analizador de tráfico de red. En él se valora la ventana deslizante del sistema que permite enlaces de alta velocidad, así como el hecho de que el lenguaje de consulta pueda ser ampliado con funciones construidas en C, permitiendo implementar algoritmos complejos, no contemplados con SQL.

Sin embargo, no están implementadas ciertas funcionalidades. Entre otras, los join entre streams y tablas solo pueden comparar elementos con el operador '='. Además hay ciertas restricciones que afectan a la parte experimental en general y particularmente a este ejemplo:

No se soportan sub-consultas.

No se soportan ventanas de salto ni rotatorias.

No está integrado el tratamiento on-line y off-line.

Estos objetivos para monitorización de red, sí que están implementados y probados en otros DSMS como Gigascope y según [1] también en STREAM.

5.4. Otros resultados experimentales con TelegraphCQ

En experimentos realizados por la Universidad de Oslo [31], se ve que el tamaño medio de los paquetes aumenta conforme aumenta la carga de red, especialmente a partir de 10 Mbits/s, y probablemente debido a que la capa TCP comienza a desfragmentar paquetes, si estos llegan a gran velocidad, para reducir el número total de los mismos.

Del mismo modo se comprueba que la degradación del rendimiento medio, está muy relacionada con la carga de red, probablemente por los motivos antes descritos, dado que la mayor caída de este rendimiento se comienza a observar entre los 6 y los 10 Mbit/s.

Como conclusión se extrae de estos experimentos que TelegraphCQ solo maneja data streams de forma correcta hasta a aproximadamente 2,5Mbits/s, siempre y cuando se tenga una velocidad constante de llegada y carga TCP de 576 bytes.

Relacionado con los experimentos anteriores en [44], se demuestra que un aumento en la velocidad de llegada de los datos, produce un aumento en el error de la desviación, especialmente hasta 400 tuplas/s a partir de donde permanece casi estable hasta las 1600 tuplas/s.

5.5. Niagara CQ. Resultados experimentales con escalabilidad

En [3] se exponen una serie de resultados experimentales con una máquina SUN con Solaris 2.6 y 1Gb de RAM. En estos experimentos se tiene en cuenta el número de consultas instaladas, el número de consultas en el grupo y el número de tuplas modificadas.

Los investigadores de NiagaraCQ esperan que una optimización incremental del grupo provea una sustancial mejora en procesamiento y escalabilidad, frente a aproximaciones desagrupadas, mostrando los beneficios de la computación compartida y evitando innecesarias invocaciones de consultas.

El prototipo tiene entre sus características principales un optimizador de grupo, un gestor de consultas continuas, un detector de eventos y un gestor de datos. Aunque por limitación del propio sistema solo permite las cláusulas de selección y *join*, sí que se plantearon como trabajo futuro incluir la agregación y el reagrupamiento dinámico

Se obtienen dos resultados muy concretos y exitosos para cumplir con los planteamientos que quiere conseguir el sistema.

En el primero se evalúa el tiempo de ejecución, en segundos, que lleva tratar un número de consultas determinado, de dos maneras distintas aunque en ambas para modificar un total de 1000 filas (C=1000). En la primera manera se plantea que sobre un total de N consultas instaladas, se lanzan todas, siendo F las consultas lanzadas, por tanto F=N. Se obtiene un incremento gradual desde 0 segundos para 0 consultas hasta unos 300 segundos para 10.000 consultas. La segunda forma de este experimento fija el número de consultas lanzadas a F=100. En este caso los resultados son aún más positivos y cercanos a obtener los resultados inmediatamente, dado que el número de consultas ejecutadas es muy pequeño comparado con N, aun cuando N=10000, el tiempo de ejecución es solo función de F.

Como en los sistemas sin agrupación T es proporcional a C porque el operador de selección de todas las consultas tiene que ser ejecutado siempre. Sin embargo, en este sistema agrupado, T o tiempo de ejecución no es sensible a F porque es un pequeño porcentaje del total de consultas C.

En el segundo experimento se evalúa el tiempo de ejecución conforme aumenta el número de tuplas modificadas, siendo F=N. Mientras que en un sistema no agrupado crece regularmente el T con el número de tuplas desde unos 80 s para 100 tuplas hasta 750s para 2000 tuplas, el sistema agrupado se mantiene constante en torno a 20s de 100 a 2000 tuplas.

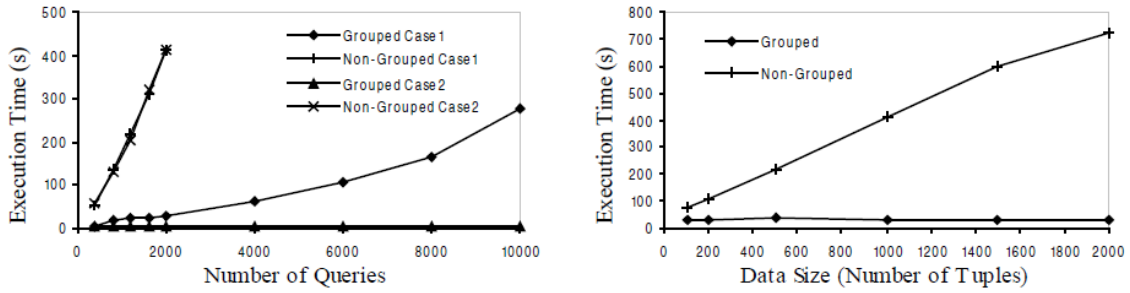

Figura 5.2. Resultados experimentales de NiagaraCQ. Extraídos de [3]

5.6. TinyDB. Procesamiento en redes de sensores

Es considerado un pequeño DSMS para redes de sensores inalámbricas, por ello los resultados experimentales ejecutados hasta la fecha están orientados a la ejecución de consultas y minimización de los problemas propios de este tipo de redes.

Mediante un API en Java corriendo sobre TinyOS y con el lenguaje TinySQL, se realizan una serie de experimentos centrados en el consumo energético, precisamente uno de los puntos fundamentales a tener en cuenta a la hora de diseñar y evaluar una red de sensores como se ha visto en este trabajo.

En un primer resultado se observa que el consumo energético está directamente relacionado con la duración del evento y es directamente proporcional al número de eventos por segundo.

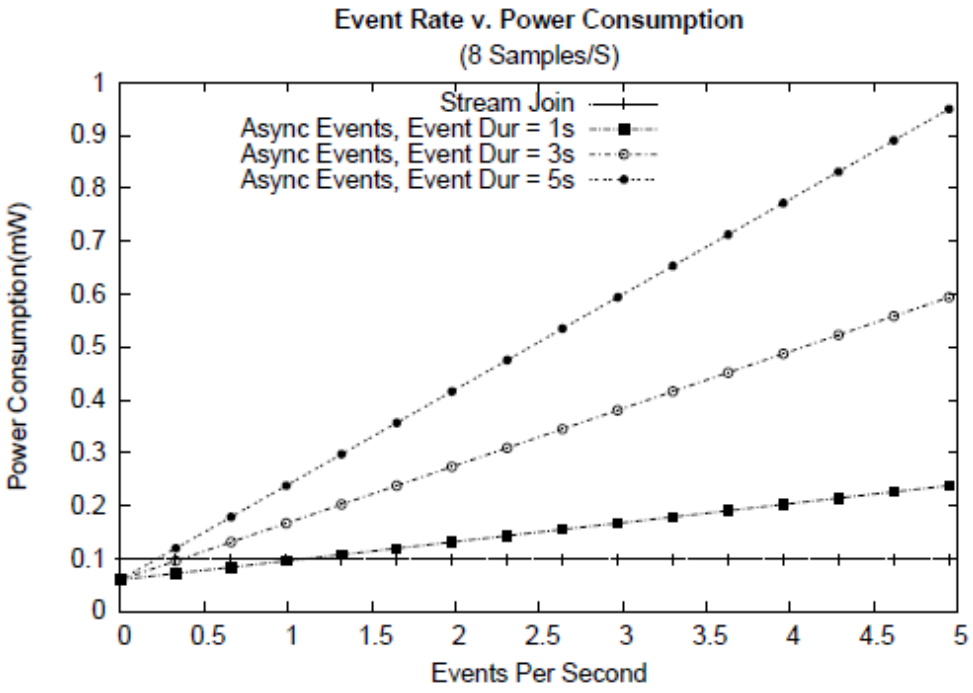

Figura 5.3. Coste de procesar consultas basadas en eventos como eventos asíncronos vs joins Extraída de. [28]

Otros trabajos publicados como [45] ahondan en los estudios sobre el consumo energético, además de otros conceptos como es la estabilidad y fiabilidad del envío de paquetes en diferentes condiciones.

Se realizan los experimentos con sensores hardware Mica2 y para evaluar un mayor número de estos se realizan más pruebas sobre el entorno de simulación TOSSIM, basándose en TinyOS. Estas pruebas se basan en una estructura lineal de sensores, no en una red propiamente dicha aunque sirve para evaluar estos parámetros correctamente.

En cuanto a la distancia entre los sensores, se observa que en días soleados se reciben correctamente el 100% de los paquetes hasta 50m y que la calidad de la señal cae hasta el

30% de los paquetes cuando los sensores se separan hasta 110m. Sin embargo, en días de lluvia, el 100% de los paquetes llega con una separación de 10m entre sensores y no se recibe nada si estos están separados 50m.

Además se observa que si los nodos están separados 80m la distancia se limita a 5 saltos, momento en el cual se pierden los paquetes. Sin embargo si la distancia se limita a 40m la transmisión de paquetes es correcta al 100% del primer al último nodo.

El consumo energético se mide con 2 pilas de1.5V en cada sensor. Se observa que el nodo más alejado del receptor final conservará mucho más la energía dado que sólo debe enviar sus datos, mientras que el nodo más cercano al receptor debe transmitir a este la información propia y la del resto de nodos. En este caso, el sistema seguirá en funcionamiento tanto tiempo como dure la pila que antes se agote, en este caso la del sensor más cercano al receptor. Su autonomía, teniendo en cuenta que por debajo de 2V dejan de transmitir, es de 2,25 días habiendo hecho 150 lecturas cada nodo.

En un entorno simulado con TOSSIM, se observa que con una separación entre nodos de 33m solo se reciben el 100% de los paquetes hasta el 2º nodo, y llegarán sólo el 10% del 10º nodo.

Los tiempo de respuesta también son evaluados en el simulador y con sensores hardware Mica2, obteniendo un aumento gradual del mismo desde los 25s con un nodo hasta los 40s con 6 nodos en un entorno real, y 160s con un nodo en TOSSIM hasta 400s con 10 nodos, aunque se detecta muy poco incremento a partir de 25 nodos rondando los 500s.

Por todo ello se concluye que es un sistema poco escalable, en términos de fiabilidad, en una topología lineal.

En términos de corrección se han hecho estudios para evaluar TinyDB, como los mostrados en [46], donde se demuestra que TinyDB, antes de depurar la API de Java produce resultados erróneos para consultas agregadas por defecto. Estos errores consisten en que sólo se muestran resultados el 80% de las veces y que entre el 20 y el 30% de las ejecuciones se pierden paquetes a la hora de procesarlos.

En cuanto a la aportación que se realiza cabe destacar el hecho de que se implemente la ejecución de sentencias con las cláusulas *having* y *group by*.

5.7. Exp. Borealis. Distribución carga y tolerancia fallos

Borealis y los experimentos ejecutados con este sistema en [13] se centran en la distribución de carga, principalmente ejecutados con el simulador planteado con la librería CSIM [47], teniendo en cuenta que el coste de ejecución de cada operador es de 0.1s, un total de 20 nodos, un ancho de banda de 100Mbps, una ventana estática de 10s en la que se recogen 10 muestras y un periodo de distribución de carga de 1s.

Con estos datos se observa que el ratio de latencia aumenta especialmente a partir del 90% de nivel de carga pero que la desviación media de la carga estándar apenas sufre un ligero incremento.

A continuación en el estudio se observan las variaciones y cambios a los que comúnmente son sensibles estos sistemas. Estos son: el número de nodos, la media de operadores en cada nodo, el tamaño de las ventanas estáticas, el número de muestras tomadas en cada ventana y las fluctuaciones en la entrada de los datos. Todo ello con un nivel de carga del sistema del 90%.

Como conclusión se extrae que los algoritmos evaluados, basados en correlación, no son muy sensibles a estos cambios, incluso cuando el tamaño de las ventanas es solo la mitad del periodo de fluctuación de carga que en estos experimentos se fija en 20.

Por otro lado la tolerancia a fallos se estudia en [15] donde se plantea una aproximación basada en la replicación. En esta se contemplan fallos de nodos, de red y de particiones de red. Gracias modelo planteado que distingue las tuplas estables de las provisionales, resultantes de procesar parcialmente los datos de entrada que pueden ser corregidos más tarde. Asumen que favorece la disponibilidad garantizando la consistencia.

Para fallos cortos, los nodos pueden eludir la inconsistencia bloqueando y buscando un nodo vecino estable. En el caso de los fallos más largos, será necesario que los nodos procesen nuevos datos de entrada tanto mientras dure el fallo como en la estabilización para asegurar la disponibilidad requerida. El sistema con creando un punto de restauración y rehaciendo desde el mismo, es más rápido que el de deshacer y rehacer para conseguir una rápida reconciliación entre nodos en caso de fallo.

Este sistema, evidentemente es mucho más útil y efectivo en las aplicaciones que requieren de menos retrasos aunque los resultados sean aproximados, pero permitiendo, eventualmente, ver los streams correctos de salida. Se indica expresamente como requerimiento del manejo de fallos en estos sistemas.

5.8. Procesamiento eventos complejos. SASE-TelegraphCQ

Existen una serie de publicaciones con experimentos exhaustivos para valorar el sistema como los publicados en [21] que están basados en un prototipo basado en Java, corriendo en un PIII a 1.4 GHz y 1.5GB de RAM ejecutando un Sun J2RE 1.5 en Fedora Linux 2.6.12. Se contemplan 20 tipos de eventos, 5 atributos pro evento y un tamaño de dominio de entre 10 y 10.000 valores permitidos en cada atributo.

En el primer experimento se evalúa la variación en el tamaño del dominio, viéndose que al aumentar este, el número de eventos por segundo gestionados mejora, debido a que el número de resultados de las consultas desciende.

Sin embargo al aumentar el tamaño de la ventana se observa una caída en el número de eventos, como se ve en el segundo experimento.

En este mismo trabajo se ha visto que el rendimiento normalizado cae dramáticamente con la longitud de la secuencia, aunque comparado con TelegraphCQ escala mucho mejor, permaneciendo en todo momento los valores devueltos por SASE por encima, y cada vez con mayor diferencia, respecto a los de TelegraphCQ.

También se dispone de una comparativa de rendimiento entre TelegraphCQ y SASE, positiva para este último, viendo cómo varía la selectividad de consulta. Una vez mostrados los resultados se concluye que un procesador de stream relacional como TelegraphCQ no está diseñado ni optimizado para procesamiento de eventos complejos. Sin embargo SASE emplea operadores nativos para manejar secuencias de eventos y planes altamente optimizados para reducir el tamaño de los resultados intermedios. Esto entre otras cosas, permite afirmar que las técnicas de este sistema son efectivas y escalables para el procesamiento de eventos complejos.

6. Conclusiones

Los dominios de aplicación de los Data Stream Management Systems son muy amplios, más aún que las distintas soluciones presentadas en este trabajo. Se ha ejecutado una presentación de cada sistema y una evaluación cualitativa donde se ve los distintos tipos de criterio existentes en cada sistema así como las posibilidades y las limitaciones que ofrece cada uno de ellos.

De una manera sencilla e intuitiva, un investigador o desarrollador que quiera plantearse un trabajar con DSMS, puede sin duda, teniendo claro cuál será su vía de trabajo, extraer conclusiones rápidas que le permita elegir el DSMS que más se ajuste a sus necesidades.

No obstante con los diferentes experimentos y la orientación que los desarrolladores dan a sus respectivos sistemas y que se han expuesto en este trabajo se extraen conclusiones claras sobre los sistemas que actualmente resultan más adecuados según el dominio de aplicación sobre el que se quieran aplicar.

En primer lugar se debe hacer una separación entre los sistemas de propósito general y aquellos de aplicación en entornos más restringidos.

Los sistemas de propósito general estudiados en este trabajo son STREAM y TelegraphCQ, aunque precisamente para hacer una comparativa entre ellos se debe elegir un entorno concreto, como en este caso es el de la medición de consumo energético. Aunque el entorno sea concreto se pueden extraer conclusiones generales.

Mientras TelegraphCQ resulta más sencillo y flexible a la hora de escribir las consultas y reutilizar los datos conseguidos, STREAM dispone de una GUI que permite monitorizar en tiempo real el estado del sistema. Si esta última característica no es determinante en el trabajo que se quiera desarrollar, TelegraphCQ sería el DSMS de propósito general más versátil.

No obstante ninguno de estos sistemas tendría expresamente integrada una funcionalidad para integrar la medición automática de datos necesaria en redes de sensores. Si bien es cierto que con TelegraphCQ se han realizado desarrollos que permitieran monitorizar resultados devueltos por sensores y monitorización de tráfico de red, demostrándose que los dos hándicap indicados a priori sobre el sistema pueden ser salvables en cierto modo.

Un problema importante a tener en cuenta es la limitación en cuanto al ratio de datos recibidos para evitar que el sistema genere errores de pérdida de datos y degradación de la desviación de estos.

En este campo precisamente en dónde entra la solución de TinyDB que resulta muy útil y sencilla, para realizar consultas y de aplicar en entornos de redes de sensores, haciendo mención expresa a dispositivos móviles del tipo PDA. Además es una solución integral dado que aporta su propio sistema operativo, su propio lenguaje de consultas y su propio simulador ejecutable en Linux y Windows, aunque resulte un poco restringido a la hora de plantear soluciones externas.

En cuanto a escalabilidad los resultados más concretos y positivos son los mostrados por NiagaraCQ, sobretodo en cuanto al número de consultas concurrentes a ejecutar y el número de tuplas afectadas, muy orientado al entorno de Internet, que será a nivel mundial el que más necesidad tenga de esta característica. No obstante debe tenerse en cuenta para aquellas redes de sensores dónde se prevea un incremento exponencial en el número de sensores y por tanto en el número de consultas y datos que se devolverán.

La tolerancia a fallos tratada expresamente por Borealis no es del todo fiable, dado que expresamente el sistema se centra en un suministro de datos con poco retardo aunque los datos suministrados no sean exactos. Este es un punto que puede resultar crítico en muchos sistemas y debe tenerse en cuenta antes de elegir Borealis como DSMS.

Uno de los últimos sistemas desarrollados es SASE, centrado especialmente en el procesamiento de eventos complejos. En concreto además de los resultados propios del sistema se compara su rendimiento en idénticas condiciones con TelegraphCQ saliendo muy beneficiado en la comparativa.

En concreto los aspectos que mejora SASE respecto a TelegraphCQ, que recordemos había salido valorado positivamente de la comparativa con STREAM, son el rendimiento normalizado, la escalabilidad que se intuye mejor también que la existente en NiagaraCQ. Probablemente el motivo de estos resultados sea que ni TelegraphCQ ni NiagaraCQ están expresamente diseñados para eventos complejos como sí que lo está SASE.

7. Trabajo futuro

De este trabajo pueden generarse dos líneas de investigaciones futuras e independientes. La primera deriva directamente del hecho de que en todos estos experimentos hay un punto a tener en cuenta y es que sus resultados están obtenidos en entornos y condiciones diferentes.

El diseño de un benchmark, para la evaluación experimental exacta de los DSMS existentes, debería ser objeto de una investigación más extensa. En esta se encontrarán los típicos problemas de diseño e implementación de benchmarks como es la definición de los puntos a evaluar, si bien estos ya han sido mostrados en este trabajo. Principalmente: tiempo de respuesta, precisión, escalabilidad y rendimiento, así como una evaluación cualitativa de cómo de sencillo resulta definir y ejecutar las mismas consultas en uno y otro entorno.

Sin embargo habrá otros propios del conjunto de DSMS existentes y los distintos ámbitos de aplicación. Precisamente por estar diseñados para dominios diferentes con la intención de solucionar problemas distintos, este Benchmark debe plantearse como un diseño a medio o largo plazo, teniendo en cuenta no sólo un entorno de aplicación si no varios. Algunos de estos se han visto en este trabajo, como consumo energético, enrutamiento de datos, monitorización de redes, monitorización de sistemas, tipos de consultas, rendimiento con cargas variables del sistema, tamaño de datos admitidos, errores y desviaciones, etc.

Además de esta propuesta de trabajo futuro, habría una segunda línea de investigación que englobaría a unos sistemas solo parcialmente mencionados en el trabajo, como es el caso de QueryMesh que propone un procesamiento adaptativo de consultas por varias rutas como en CAPE. Actualmente este sistema junto a Eddies y CBR forman una línea de investigación independiente y sin duda el estado en el que se encuentran las propuestas actuales puede ser objeto de un estudio futuro más exhaustivo en sí mismo.

8. Bibliografía

[1] Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J., "Models and Issues in Data Stream Systems," in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*. Madison, USA: ACM, 2002, pp. 1-16.

[2] Golab, Lukasz; Özsu, M. Tamer, "Issues in data stream management," in *Proceedings of the SIGMOD'03*, San Diego, CA, 2003.

[3] Chen, Jianjun; DeWitt, David J.; Tian, Feng; Wang, Yuan, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," in *In Proceedings of the SIGMOD 2000*, Dallas, USA, pp. 379-390.

[4] Naughton, Jeffrey; DeWitt, David; Maier, David, "The Niagara Internet Query System," Wisconsin-Madison, Overview paper 2000.

[5] Wang, Song, State-Slice: A New Stream Query Optimization Paradigm for Multi-query and Distributed Processing, 2008.

[6] Abadi, Daniel J.; Carney, Donald P.; Cetintemel, Ugur; Cherniack, Mitch Frederic; Convey, Christian; Erwin, C.; Gálvez, Eduardo; Hatoun, M.; Maskey, Anurag S.; Rasin, Alexander; Singer, A.; Stonebraker, Michael Ralph; Tatbul, Nesime; Xing, Ying; Yan, Rongguo; Zdonik, Stanley B., "Aurora - A Data Stream Management System," in *In Proceedings of the SIGMOD 2003*, San Diego, USA, pp. 666-666.

[7] Cranor, Charles D.; Johnson, Theodore J.; Spatscheck, Oliver; Shkapenyuk, Vladislav, "Gigascope: A Stream Database for Network Applications," in *In Proceedings of the SIGMOD 2003*, San Diego, USA, pp. 647-651.

[8] Hammad, Moustafa A.; Mokbel, Mohamed F.; Ali, Mohamed H.; Aref, Walid G.; Catlin, Ann C.; Elmagarmid, Ahmed K.; Elmagarmid, Ahmed K.; Elfeky, Mohamed G.; Ghanem, Thanaa M.; Gwadera, Robert; Ilyas, Ihab F.; Marzouk, Mirette; Xiong, Xiaopeng, "Nile: A Query Processing Engine for Data Streams," in *Proceedings in the 20th International Conference on Data Engineering (ICDE)*, Boston, 2004, pp. 851-852.

[9] Ali, Mohamed H.; Aref, Walid G.; Bose, R.; Elmagarmid, Ahmed K.; Helal, A.; Kamel, I.; Mokbel, M. F., "NILE-PDT: A Phenomenon Detection and Tracking Framework for Data Stream Management Systems," in *In Proceedings of the 31st International Conference on VLDB*, Trondheim, Norway, 2005, pp. 1295 -1298.

[10] Elmongui, H. G.; Xiaopeng, Xion; Xiaoyong, Chai; Aref, W.G.;, "PLACE: A Distributed Spatio-Temporal Data Stream Management System for Moving Objects," in *In Proceedings of the 8th International Conference on Mobile Data Management*, Mannheim, Germany, 2007, pp. 44-51.

[11] Amini, Lisa D.; Andrade, Henrique C. M.; Bhagwan, Ranjita; Eskesen, Frank; King, Richard P.; Selo, Philippe; Park, Yoonho; Venkatramani, Chitra, "SPC: A distributed, scalable platform for data mining," in *In Proceedings of the 4th International Workshop on Data Mining Standards, Services and Platforms*, Philadelphia, USA, 2006, pp. 27-37.

[12] Arasu, Arvind; Babcock, Brian; Babu, Shivnath; Datar, Mayur; Ito, Keith; Nishizawa, Itaru; Rosenstein, Justin; Widom, Jennifer, "STREAM: The Stanford Data Stream Management System," in *In Proceedings of the SIGMOD 2003*, San Diego, USA, pp. 644-665.

[13] Xing, Ying; Zdonik, Stan; Hwang, Jeong-Hyon, "Dynamic Load Distribution in the Borealis Stream Processor," in *In Proceedings of the 21st ICDE 2005*, Tokyo, Japan, pp. 791-802.

[14] Tatbul, Nesime; Ahmad, Yanif; Cetintemel, Ugur; Hwang, Jeong-Hyon; Xing, Ying; Zdonik, Stan, "Load Management and High Availability in the Borealis Distributed Stream Processing Engine," in *Lecture Notes in Computer Science*., 2008, vol. 4540/2008, pp. 66-85.

[15] Balazinska, Magdalena; Balakrishnan, Hari; Madden, Samuel; Stonebraker, Michael, "Fault Tolerance in the Borealis Distributed Stream Processing System," *ACM Transactions on Database Systems*, vol. 33, no. 1, pp. 1-44, Marzo 2008.

[16] Abadi, Daniel J.; Ahmad, Yanif; Balazinska, Magdalena; Cetintemel, Ugur; Cherniack, Mitch; Hwang, Jeong-Hyon; Lindner, Wolfgang; Maskey, Anurag S.; Rasin, Alexander; Ryvkina, Esther; Tatbul, Nesime; Xing, Ying; Zdonik, Stan;, "The Design of the Borealis Stream Processing Engine," in *In Proceedings of the 2nd Biennial CIDR 2005*, Asilomar, USA.

[17] Demers, Alan; Gehrke, Johannes; Rajaraman, Rajmohan; Trigoni, Niki; Yao, Yong, "The Cougar Project: A Work-In-Progress Report," *SIGMOD Rec.*, vol. 32, no. 4, pp. 53-59, Diciembre 2003.

[18] Nehme, Rimma V.; Works, Karen; Rundensteiner, Elke A.; Bertino, Elisa , "Query Mesh: An Efficient Multi-Route Approach to Query Optimization," in *CSD TR #08-009*, 2008.

[19] Nehme, Rimma V.; Works, Karen E.; Rundensteiner, Elke Angelika; Bertino, E., "Query mesh: multi-route query processing technology," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1530-1533, Agosto 2009.

[20] Nehme, Rimma V.; Rundensteiner, Elke A.; Bertino, Elisa, "Self-tuning query mesh for adaptive multi-route query processing," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. San Petesburgo, Rusia: ACM, 2009, pp. 803-814.

[21] Wu, Eugene; Diao, Yanlei; Rizvi, Shariq, "High-Performance Complex Event Processing over Streams," in *Proceedings of the 2006 ACM SIGMOD*. Chicago, USA: ACM, 2006, pp. 407-418.

[22] Gyllstrom, Daniel; Wu, Eugene; Chae, Hee-Jin; Diao, Yanlei; Stahlberg, Patrick; Anderson, Gordon, "SASE: Complex Event Processing over Streams," in *In Proceedings of the 3rd Biennial CIDR 2007*, Asilomar, USA.

[23] Agrawal, Jagrati; Diao, Yanlei; Gyllstrom, Daniel; Immerman, Neil, "Efficient Pattern Matching over Event Streams," in *Proceedings of the 2008 ACM SIGMOD*. Vancouver, Canadá: ACM, pp. 147-160.

[24] Diao, Yanlei; Immerman, Nei; Gyllstrom, Daniel, "SASE+: An Agile Language for Kleene Closure over Event Streams," in *UMass Technical Report*, Massachusetts, USA, 2007.

[25] Gyllstrom, Daniel; Agrawal, Jagrati; Diao, Yanlei; Immerman, Neil, "On Supporting Kleene Closure over Event Streams," in *In Proceedings of the 24th ICDE 2008*, Cancún, México, pp. 1391-1393.

[26] Chandrasekaran, Sirish; Cooper, Owen; Deshpande, Amol; Franklin, Michael J.;

Hellerstein, Joseph M.; Hong, Wei; Krishnamurthy, Sailesh; Madden, Sam; Raman, Vijayshankar; Reiss, Fred; Shah, Mehul, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *In Proceedings of the 1st Biennial CIDR 2003*, Asilomar, USA.

[27] Madden,Sam; Hellerstein,Joe ;Hong,Wei, "TinyDB: In-Network Query Processing in TinyOS," , Berckeley, USA, 2002.

[28] Madden, Samuel R.; Franklin, Michael J.; Hellerstein, Joseph M.; Hong, Wei, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122-173, Marzo 2005.

[29] Stonebraker, Michael; Cetintemel, Ugur; Zdonik, Stan, "The 8 Requirements of Real-Time Stream Processing," *SIGMOD Rec.*, vol. 34, no. 4, pp. 42-47, Diciembre 2005.

[30] Golab, Lukasz; Özsu, M. Tamer, "Issues in data stream management," *SIGMOD Rec.*, vol. 32, no. 2, pp. 5-14, Junio 2003.

[31] Soberg, Jarle, Design, Implementation and Evaluation of Network Monitoring Task with the TelgraphCQ DSMS, 2006, Master's Thesis. University of Oslo.

[32] Gehrke, Johannes; Korn, Flip; Srivastava, Divesh, "On computing correlated aggregates over continual data streams," *SIGMOD Rec.*, vol. 30, no. 2, pp. 13-24, Mayo 2001.

[33] L. M. S. C. of the IEEE Computer Society. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification., 1999.

[34] Arasu, Arvind; Babu, Shivnath; Widom, Jennifer, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121-142, Junio 2006.

[35] Stonebraker, M.; Rowe, L., "The design of POSTGRES," *SIGMOD Rec.*, vol. 15, no. 2, pp. 340-355, Junio 1986.

[36] Stonebraker, M.; Hanson, E.; Hong, C. H., "The design of the POSTGRES rules system," *SIGMOD Rec.*, 1987.

[37] Stonebraker, Michael; Rowe, Lawrence A., "The design of the POSTGRES storage system," *SIGMOD Rec.*, vol. 15, no. 2, pp. 340-355, Junio 1986.

[38] Zdonik, S.; Stonebraker, M.; Cherniack, M.; Cetintemel, U.; Balazinska, M., "The Aurora and Medusa projects," *IEEE Data Engineering Bulletin*, vol. 26, 2003.

[39] Avnur, Ron; Hellerstein, Joseph M., "Eddies: Continuously adaptative query processing," *SIGMOD Rec.*, vol. 29, no. 2, pp. 261-272, Mayo 2000.

[40] Bizarro, P., "Content-based routing: Different plans for different data," in *Proceedings of the 31st international conference on VLDB*. Trondheim, Noruega: VLDB Endowment, 2005, pp. 757–768.

[41] Rundensteiner, E.A., "Cape: Continuous query engine with heterogeneous-grained adaptivity," in *Proceedings of the 30th international conference on VLDB*. Toronto, Canadá, 2004, vol. 30, pp. 1353–1356.

[42] Abdessalem, T.; Chiky, R.; Hébrail, G.; Vitti, J.L., "Using Data Stream Management Systems to analyze Electric Power Consumption Data," in *International Workshop on Data Stream Analysis (WDSA)*, Caserta, Italia, 2007.

[43] Plagemann, Thomas; Goebel, Vera; Bergamini, Andrea; Tolu, Giacomo; Urvoy-Keller,

Guillaume; Biersack, Ernst, "Using Data Stream Management Systems for Traffic Analysis – A Case Study –," in *Passive and Active Network Measurement*, Chadi Barakat and Ian Pratt, Eds.: Springer Berlin / Heidelberg, 2004, vol. 3015, pp. 215-226.

[44] Reiss, Frederick; Hellerstein, Joseph M., "Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ," in *In Proceedings of the 21st ICDE 2005*, Tokyo, Japón, pp. 155-156.

[45] Gumbo, Sibukele, Development of a web-based interface for a wireless sensor network monitoring system, 2007, Master's Thesis.

[46] Kofoed, Leif Morten, Enhancing sensor network programming: Extending TinyDB with HAVING and aggregation, and investigating TinyDB reliability, 2007, Master's Thesis.

[47] Mesquite Software, Inc.. CSIM 18 Simulation Engine. [Online]. http://www.mesquite.com

[48] Chakeres, Ian D.; Belding-Royer, Elizabeth M.;, "AODV Routing Protocol Implementation Design," in *International Conference on Distributed Computing Systems Workshops in the 24th ICDCSW 2004*, Tokyo, Japón, 2004, pp. 698-703.

[49] Bennet, F.; Clarke, D.; Evans, J.; Hopper, A.; Jones, A.; Leask, D.;, "Piconet: Embedded Mobile Networking," in *IEEE Personal Communications*, 1997, pp. 8-15.

[50] Babcock, Brian; Babu, Shivnath; Datar, Mayur; Motwani, Rajeev; Widom, Jennifer, "Models and issues in data stream systems," in *In Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*, Madison, USA, 2002, pp. 1-16.

AGRADECIMIENTOS.

Quiero agradecer su dedicación y esfuerzo al Dr. Sergio Ilarri, director de esta tesis, sin el cual no habría llegado a buen puerto.

No quiero olvidarme de agradecer su trabajo al profesorado del Máster en Ingeniería de Sistemas e Informática, especialmente a aquellos de los que he tenido el placer de recibir clase.

Gracias a mis padres y a mi hermano, por su apoyo constante y su paciencia, y a mi novia Pili, por estar siempre ahí.

RESUMEN DE LA EXPERIENCIA PERSONAL

Este es el primer trabajo de investigación serio que realizo, hasta la fecha todos los proyectos desarrollados habían sido a nivel empresarial incluido el PFC, y sin duda me he enfrentado a duros, inesperados y desconocidos problemas durante su desarrollo.

Esto me ha hecho invertir muchas horas documentándome, aprendiendo a redactar, sintetizar y en general, todo lo significa investigar un tema tan extenso como es el de los Data Stream Management Systems.

La propuesta de tesis fue definida, junto a mi director el Dr. Sergio Ilarri en diciembre de 2010, y desde entonces muchas horas han sido invertidas en el trabajo. Sin llevar una cuenta exhaustiva de las mismas el número de horas sin duda está por encima de las 700.

La principal dificultad que me he encontrado ha sido luchar contra mi total falta de experiencia en el desarrollo de un trabajo de investigación y su posterior documentación, problema al que me he enfrentado a diario de diversas formas en todos y cada uno de los puntos que he tratado y de los que se compone este documento.

Además de lo anterior, por supuesto, manejar la cantidad de información recogida y generada para entender y analizar cada uno de los sistemas propuestos, ha sido una ardua tarea.

Una vez finalizado este documento compruebo que no es el final sino el principio de una extensa labor de investigación, que se irá afianzando en forma de publicaciones de artículos y nuevas líneas de investigación independientes, a partir de ahora.

A pesar de todo, con mayor o menor éxito, he adquirido una experiencia, unos conocimientos y una metodología de trabajo que desconocía hasta hace un año y con los que cada vez me encuentro más a gusto trabajando.

Desde luego lo aprendido, no solo en el desarrollo de la tesis gracias al Dr. Sergio Ilarri, sino también el resto aptitudes conseguidas a lo largo de todas las asignaturas del máster con el profesorado que las ha impartido, hacen que la valoración personal de lo que me ha aportado este último año sea, sin dudarlo, absolutamente positiva.