

Gestión de Datos para Robótica

T4b - HStreamDB

Álvaro Vázquez Álvarez
Departamento de Electrónica e Computación

✉ alvaro.vazquez@usc.es

📍 Pabellón III - Despacho 4

Curso 2023-2024

Tabla de contenidos

- Introducción. Conceptos básicos
- Plataforma HStreamDB
 - HStream Server (HSQL)
 - HStream Storage (HStorage)
 - Instalación y despliegue con Docker y Kubernetes
- Cliente CLI SQL
- API HStreamDB para Python (hstreamdb-py):
 - Instalación y ejemplos.
 - Operaciones básicas: creación, borrado y consulta de streams
 - Operaciones de lectura y escritura de datos en Streams
- Referencias

Introducción

HStreamDB: Sistema Gestor de Bases de Datos en streaming (open source/comercial)



CARACTERÍSTICAS

- Optimizado para almacenar, procesar, proporcionar acceso y distribuir grandes volúmenes de flujo de datos (data streams) en tiempo-real de fuentes como dispositivos IoT.
- Altamente escalable: cluster compuesto de nodos HStreamDB.
- Usa el estándar SQL + extensiones para data streaming como lenguaje principal para la interfaz de consulta.
- API para crear interfaces de usuario implementada en clientes Java, Go y Python.
- **Objetivo:** simplificar la operación y gestión de flujos de datos y el desarrollo de aplicaciones real-time.

Conceptos básicos (I)

REGISTRO

Unidad mínima de información que puede contener una cantidad arbitraria de datos de usuarios y es inmutable. Cada registro tiene un **recordID** único por Stream. Todos los **registros residen en Streams**.

STREAM

Un **Stream** es básicamente un conjunto de datos incremental ilimitado (no se pueden eliminar datos del Stream una vez agregados). Puede contener múltiples **Shards** (fragmentos) y cada Shard del Stream puede estar localizado en un nodo HStreamDB distinto. Un Stream está caracterizado por el número de réplicas y el tiempo de retención de los datos en el Stream.

SUSCRIPCIÓN

Los **clientes** pueden acceder a los datos más recientes de los Streams en tiempo real mediante las **suscripciones**. Se pueden crear múltiples suscripciones independientes al mismo Stream. Cada suscripción puede realizar el seguimiento y guardar el progreso del procesamiento de registros de los clientes. El modo de suscripción por defecto es basado en Shards: todos los registros de un Shard serán enviados al mismo cliente, y Shards diferentes pueden ser asignados a clientes diferentes.

Conceptos básicos (II)

CONSULTA

A diferencia de las consultas en las bases de datos tradicionales que operan sobre conjuntos de datos estáticos, devuelven un conjunto de resultados limitados y terminan su ejecución de forma inmediata, las consultas en HStream operan sobre Streams de datos de tamaño indeterminado, actualizando los resultados a medida que el origen de los Streams cambia y se mantienen en ejecución hasta que el usuario los detiene explícitamente. Por defecto, una consulta escribirá los resultados de su procesamiento continuamente a un nuevo Stream. Los clientes pueden suscribirse al nuevo Stream para obtener actualizaciones en tiempo real de los resultados de procesamiento.

VISTA MATERIALIZADA

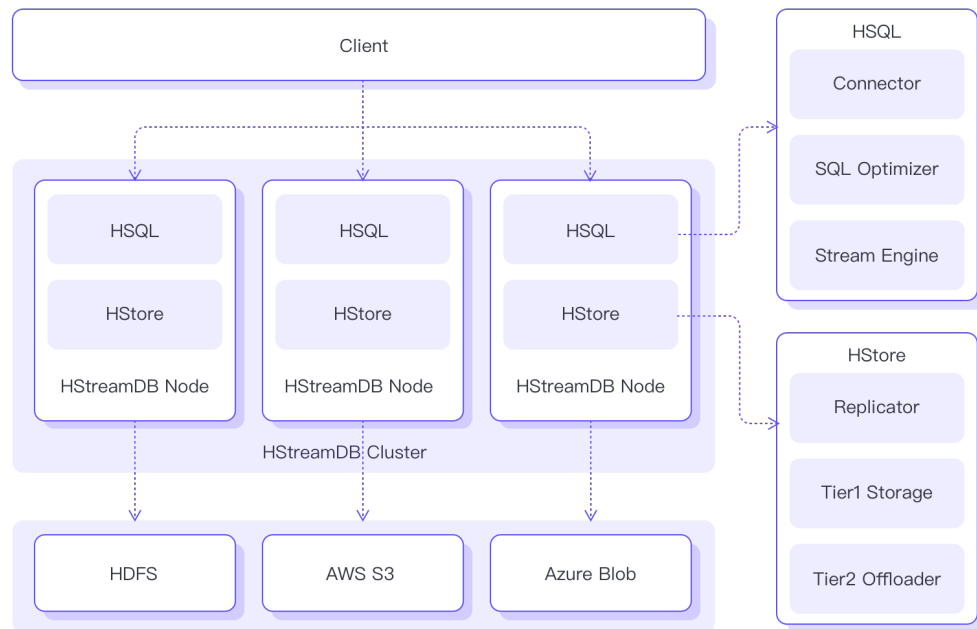
Las consultas también pueden ser usadas para crear vistas materializadas. A diferencia de los Streams que almacenan registros solo en modo de no eliminación, las vistas materializadas son más similares a las tablas de las BD relacionales, de forma que se pueden consultar directamente y obtener los resultados actualizados inmediatamente. Los resultados guardados en vistas materializadas son actualizados automáticamente en tiempo real con los cambios en los Streams de origen.

CONECTOR

Los conectores son responsables de la integración de los datos en streaming entre HStream y los sistemas externos. Existen conectores para diferentes sistemas como PostgreSQL, MongoDB, etc...

Arquitectura HStreamDB

- Un **nodo HStreamDB** consta de dos componentes principales:
 - Servidor HStream (HSQL/HServer)
 - Almacenamiento HStream (HStore)
- Cada **cluster HStreamDB** está compuesto de varios **nodos HStreamDB peer-to-peer**.
- Los **clientes** se pueden conectar a cualquier nodo HStreamDB del cluster y realizar procesamiento y análisis en streaming a través del lenguaje estándar SQL.



Servidor HStream (HSQL/HServer)

Objetivo: analizar los datos a medida que se reciben en tiempo real y entregar los resultados del análisis o procesamiento de datos a los clientes (consumidores) con baja latencia.

En un SGBD convencional los datos solo son analizados cuando el cliente envía una solicitud (procesamiento batch o por lotes).

Servidor HStream (HSQL/HServer)

ACCESO Y MANIPULACIÓN DE DATOS

- Todos los **registros de datos añadidos** a la BD se agregan a un objeto inmutable llamado **STREAM**.
- Puede haber varios **Streams** actuando **simultaneamente** en una misma BD.
- El servidor brinda **acceso de baja latencia** a los análisis de los **datos más recientes** en los Streams: a medida que llegan los flujos de datos, HStream actualiza de forma incremental en tiempo real las vistas creadas en memoria.

DISTRIBUCIÓN DE DATOS Y CONSULTAS

- Capacidad de entregar datos a múltiples **Consumers** (clientes consumidores) a través de **suscripciones**, en las que varios **Consumers** pueden leer datos en tiempo real desde un solo **Stream** a medida que los **Producers** (clientes productores) agregan registros de datos.
- Las **suscripciones** entregan los datos al cliente una vez que ya han sido incorporados al SGBD.
- Las consultas se tratan como tareas que obtienen datos de flujos y producen resultados continuamente a medida que se actualizan los flujos.

Almacenamiento HStream (HStore)

ARQUITECTURA DE ALMACENAMIENTO

- Base de Datos orientada a disco que utiliza el motor de almacenamiento **RocksDB**.
- El motor RocksDB permite que HStreamDB admita flujos de datos a gran escala.

MODELO DE ALMACENAMIENTO

- No implementa su propia capa de almacenamiento: se basa en RocksDB.
- Los datos se procesan y después se almacenan en el formato de archivo de clave-valor implementado en RocksDB.

Instalación y Despliegue con Docker + Kubernetes

Instalación Docker-Compose

<https://hstream.io/docs/en/latest/start/quickstart-with-docker.html#requirement>

```
1 docker-compose -f quick-start.yaml up -d
```

Instalación Kubernetes Cluster

<https://hstream.io/docs/en/latest/operation/deployment/deploy-k8s.html>

```
1 kubectl apply -k .
```

	NAME	READY	STATUS	RESTARTS	AGE
1	hstream-server-0	1/1	Running	0	6d1
2	8h				
3	hstream-server-1	1/1	Running	0	6d1
4	8h				
5	hstream-server-2	1/1	Running	0	6d1
6	8h				
7	logdevice-0	1/1	Running	0	6d1
8	8h				
9	logdevice-1	1/1	Running	0	6d1
10	8h				
11	logdevice-2	1/1	Running	0	6d1
12	8h				
	logdevice-3	1/1	Running	0	6d1
	8h				
	logdevice-admin-server-deployment-5c5fb9f8fb-27jlk	1/1	Running	0	6d1
	8h				
	zookeeper-0	1/1	Running	0	6d2
	2h				
	zookeeper-1	1/1	Running	0	10d
	zookeeper-2	1/1	Running	0	6d




Cliente CLI SQL

Interfaz SQL usa dialecto subconjunto SQL-92 con extensiones para Data Streaming

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22

```



Command

- :h To show these help info
- :q To exit command line interface
- :help [sql_operation] To show full usage of sql statement

SQL STATEMENTS:

- To create a simplest stream:
CREATE STREAM stream_name;
- To create a query select all fields from a stream:
SELECT * FROM stream_name EMIT CHANGES;
- To insert values to a stream:
INSERT INTO stream_name (field1, field2) VALUES (1, 2);

>

```

1 > INSERT INTO info (product, category) VALUES ("Apple", "Fruit");
2 Done.
3 > INSERT INTO visit (product, user, length) VALUES ("Apple", "Alice", 10);
4 Done.
5 > INSERT INTO visit (product, user, length) VALUES ("Apple", "Bob", 20);
6 Done.
7 > INSERT INTO visit (product, user, length) VALUES ("Apple", "Caleb", 10);
8 Done.

```

```

1 > SELECT * FROM info EMIT CHANGES;
2 {"product":"Apple","category":"Fruit"}

```

```

1 > SELECT product, MAX(length) AS max_len FROM visit GROUP BY product EMIT CHANGES;
2 {"product":"Apple","max_len":10.0}
3 {"product":"Apple","max_len":20.0}
4 {"product":"Apple","max_len":20.0}

```

API HStreamDB Python

Ejemplos básicos de uso de la API cliente HStreamDB para Python:

Creación de un cliente (Producer/Consumer) con conexión insegura al servidor HStreamDB
Creación, listado y borrado de Streams

<https://github.com/hstreamdb/hstreamdb-py/blob/main/examples/snippets/guides.py>

Escritura básica de datos a HStreamDB con el método append
Escritura de datos con un objeto BufferedProducer
Lectura de datos con subscripción mediante un objeto Consumer
Lectura de datos sin subscripción con el método reader

<https://hstreamdb.github.io/hstreamdb-py/examples.html>

Cliente HStreamDBClient

Cliente con conexión insegura al servidor HStreamDB

```
async hstreamdb.insecure_client(host='127.0.0.1', port=6570, url=None)
```

Crea un cliente con una conexión insegura a un cluster HStreamDB

Parámetros:

- **host** – IP del servidor HStreamDB, valor por defecto '127.0.0.1'
- **port** – puerto TCP de conexión con el servidor HStreamDB, valor por defecto 6570
- **url** – url alternativa para conectarse al cluster HStreamDB, en el formato 'hstream://mi-host'.

Nota: al proporcionar una url los args 'host' y 'port' serán ignorados.

Retorno:

Un objeto **HStreamDBClient**

Cliente HStreamDBClient

Cliente con conexión segura al servidor HStreamDB

```
async hstreamdb.secure_client(host='127.0.0.1', port=6570, url=None, is_creds_file=False, root_certificates=None, private_key=None, certificate_chain=None)
```

Crea un cliente con una conexión segura a un cluster HStreamDB

Parámetros:

- **host** – IP del servidor HStreamDB, valor por defecto '127.0.0.1'
- **port** – puerto TCP de conexión con el servidor HStreamDB, valor por defecto 6570
- **url** – url alternativa para conectarse al cluster HStreamDB, en el formato 'hstream://mi-host'. Nota: al proporcionar una url los args 'host' y 'port' serán ignorados.
- **is_creds_file** – si es 'True' el valor de los campos root_certificates, private_key, certificate_chain indican las rutas de los correspondientes ficheros de credenciales.
- **root_certificates** – el certificado raíz PEM indicado como cadena de bytes, o None para recuperarlo desde una localización por defecto indicada por el runtime gRPC.
- **private_key** – la clave privada PEM indicado como una cadena de bytes, o None si no se usa clave privada.
- **certificate_chain** – El certificado PEM indicado como cadena de bytes, o None if no se usa certificado.

Retorno:

Un objeto **HStreamDBClient**

Cliente HStreamDBClient

Métodos del objeto HStreamDBClient para gestión de Streams

async **init_cluster_info()** #Proporciona información del cluster HStreamDB

#Crea un nuevo Stream, si ya existe devuelve una referencia al Stream existente

async **create_stream**(*name*, *replication_factor*=1, *backlog*=0, *shard_count*=1)

Parámetros:

- **name** – nombre del stream
- **replication_factor**– como puede ser replicado el stream a través de los nodos del cluster.
- **backlog** – cuanto tiempo los streams HStreamDB retienen los registros después de ser añadidos (append), en segundos
- **shard_count** – número de fragmentos (shards).

async **delete_stream**(*name*, *ignore_non_exist*=False, *force*=False) #Elimina un Stream

async **list_streams**() #Lista todos los streams y devuelve un Iterator a una lista de objetos Stream

Cliente HStreamDBClient

Métodos del objeto HStreamDBClient para escritura/lectura de datos

async **append**(*name*, *payloads*, *key=None*, *compresstype=None*, *compresslevel=9*) #Agrega datos (payload) a un Stream

Parámetros:

- **name** (*str*) – nombre del stream
- **payloads** – una cadena, bytes o un json.
- **key** (*str* | *None*) – Clave del stream opcional

Retorno:

Iterator[*RecordId*] # un objeto Iterator a una lista de índices de registros creados

async **create_reader**(*stream_name*, *reader_id*, *shard_offset*, *timeout*, *shard_id=None*, *stream_key=None*) #Crea un reader para leer datos de un Stream

async **read_reader**(*reader_id*, *max_records*) #Recuperación de registros de datos de un Stream mediante un Reader

async **delete_reader**(*reader_id*) #Elimina un reader creado para leer datos de un Stream

Cliente HStreamDBClient

Métodos del objeto HStreamDBClient para Suscripción a Streams

async **create_subscription**(subscription_id, stream_name, ack_timeout=600, max_unacks=10000, offset=1)

async **list_subscriptions**()

async **does_subscription_exist**(subscription_id)

async **delete_subscription**(subscription_id, force=False)

Objetos BufferedProducer / Consumer

BufferedProducer para escritura de datos

```
class hstreamdb.BufferedProducer(flush_coro, find_stream_key_id_coro, append_callback=None, size_trigger=0, time_trigger=0, workers=1, retry_count=0, retry_max_delay=60, compress_type=None, compress_level=9)
```

```
async append(stream_name, payload, key=None)
```

```
async flush(stream_name, shard_id)
```

```
async flush_by_key(stream_name, key=None)
```

```
async flushall()
```

```
async close()
```

```
async wait()
```

```
async wait_and_close()
```

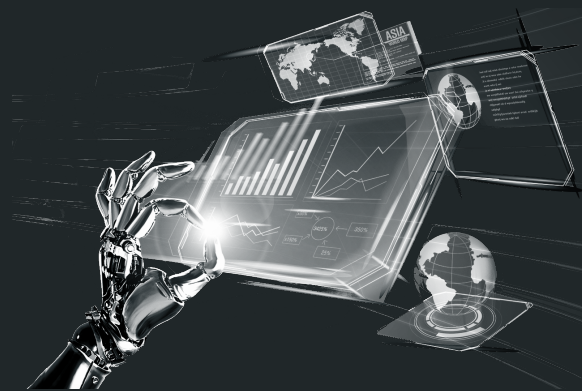
Consumer para lectura de datos

```
class hstreamdb.Consumer(name, subscription, find_stub_coro, processing_func)
```

```
async start()
```

Referencias

- HStreamDB Website. <https://hstream.io>
- Documentación HStreamDB online. <https://docs.hstream.io>
- Repositorio GitHub HStreamDB: <https://github.com/hstreamdb>
- Repositorio GitHub cliente Python. <https://github.com/hstreamdb-py>
- Documentación cliente Python. <https://hstreamdb.github.io/hstreamdb-py>



Gestión de Datos para Robótica

T4b - HStreamDB

Álvaro Vázquez Álvarez
Departamento de Electrónica e Computación

✉ alvaro.vazquez@usc.es

📍 Pabellón III - Despacho 4

Curso 2023-2024