GUIDE FOR DIRECT INTEGRATION WITH A LOW-LEVEL CMS DATABASE

МЕТОДИ ЗА ДИРЕКТНА ИНТЕГРАЦИЯ С CMS БАЗА ДАННИ НА НИСКО НИВО

Съдържание

Интегрирането на Hadoop Distributed File System (HDFS) с PostgreSQL база данни може да се подходи чрез различни методи, което позволява двупосочен поток от данни и използване на силните страни на двете системи. Тези методи варират от прости операции за импортиране / експортиране на данни до по-сложна федерация на данни и техники за репликация в реално време. Следва изчерпателен списък на подходите:

1. Пакетно прехвърляне на данни с помощта на Sqoop

Apache Sqoop е инструмент, предназначен за ефективно прехвърляне на масиви от данни между Hadoop и структурирани хранилища за данни, като например релационни бази данни. Данните могат да бъдат импортирани от PostgreSQL към HDFS и експортирани от HDFS към PostgreSQL. Sqoop използва MapReduce за импортиране и експортиране на данните, което осигурява паралелна работа, както и толерантност към грешки.

Предимства:

- Оптимизиран за прехвърляне на големи обеми от данни.
- Осигурява възможности за паралелна обработка.
- Интегрира се добре с екосистемата на Hadoop.

Недостатъци:

- Не е предназначен за интегриране на данни в реално време.
- Настройката може да бъде сложна, изискваща справедливо разбиране на екосистемата на Hadoop.
- Разчита на MapReduce, който може да добави режийни разходи в сравнение с по-новите рамки за обработка на данни.

Пример:

За да импортирате данни от PostgreSQL база данни в HDFS с помощта на Sqoop, командата може да изглежда така:

```
sqoop import \
--connect jdbc:postgresql://hostname:port/dbname \
--username dbuser \
--password dbpass \
```

```
--table tablename \
--target-dir /user/hdfs/tablename \
--m 1
```

2. Поточно поглъщане на данни с Apache Flume

Apache Flume е услуга за ефективно събиране, агрегиране и преместване на големи количества поточни данни в HDFS. Архитектурата на Flume е гъвкава и дава възможност за различни източници и дестинации. За PostgreSQL може да се напише персонализирана мивка Flume, която изтласква данни към база данни на PostgreSQL, или персонализиран източник, който чете от екземпляр на PostgreSQL.

Предимства:

- Подходящ за непрекъснато поглъщане на данни, идеален за регистрационни файлове за събития и стрийминг на данни.
- Силно адаптивни с възможност за изграждане на персонализирани компоненти.
- Може да обработва големи количества данни с висока пропускателна способност.

Недостатъци:

- Поддръжката извън кутията за PostgreSQL като мивка може да бъде ограничена, потенциално изискваща персонализирана разработка.
- Може да има по-стръмна крива на обучение за настройване на тръбопроводи за стрийминг в сравнение с партидните процеси.
- Ограничена поддръжка за транзакции или сложни трансформации на данни.

Пример:

За да предавате поточно данни от източник към HDFS и след това към PostgreSQL, може да създадете конфигурационен файл на Flume по следния начин:

```
# Дефиниране на източник, канал и мивка
agent.sources = r1
agent.channels = c1
agent.sinks = k1

# Конфигуриране на източника
```

```
agent.sources.r1.type = netcat
agent.sources.r1.bind = localhost
agent.sources.r1.port = 44444

# Конфигуриране на канала
agent.channels.c1.type = memory
agent.channels.c1.capacity = 1000
agent.channels.c1.transactionCapacity = 100

# Конфигуриране на мивката на HDFS
agent.sinks.k1.type = hdfs
agent.sinks.k1.hdfs.path = hdfs://hostname:port/path/to/dir
agent.sinks.k1.hdfs.fileType = DataStream

# Свързване на източника, мивката и канала
agent.sources.r1.channels = c1
agent.sinks.k1.channel = c1
```

3. Персонализирана картаНамаляване или задание за зареждане

Разработчиците могат да пишат персонализирани MapReduce или Spark задания, които изпълняват SQL заявки срещу PostgreSQL, за да извличат данни и да ги съхраняват в HDFS или обратно. Този метод осигурява максимален контрол върху процеса на интеграция, но изисква значителни усилия за развитие.

MapReduce е програмен модел за обработка на големи масиви от данни с паралелен, разпределен алгоритъм на клъстер, докато Spark е система за разпределена обработка с отворен код, използвана за големи натоварвания на данни.

Предимства:

- Осигурява подробен контрол върху обработката и интеграцията на данни.

- Spark заданията се изпълняват в паметта и са по-бързи от традиционните MapReduce задания.
- Мащабируем за обработка на големи обеми от данни.

Недостатъци:

- Изисква значителни усилия за развитие и опит в MapReduce или Spark.
- Интеграцията с PostgreSQL не е естествена и изисква JDBC за свързаност.
- Разработването и поддръжката могат да бъдат по-сложни в сравнение с използването на специални инструменти за интеграция.

Пример:

Работата на Spark в Scala за прехвърляне на данни от PostgreSQL към HDFS може да изглежда така:

```
val spark = SparkSession.builder.appName("PostgreSQL to HDFS").getOrCreate()

val jdbcDF = spark.read

  .format("jdbc")

  .option("url", "jdbc:postgresql://hostname:port/dbname")

  .option("dbtable", "tablename")

  .option("user", "dbuser")

  .option("password", "dbpass")

  .load()


jdbcDF.write.parquet("/user/hdfs/tablename")
```

Това задание на Spark създава DataFrame, като чете от PostgreSQL таблица и след това записва DataFrame в HDFS в паркетен формат.

4. Apache NiFi

Apache NiFi е платформа за логистика на данни, която позволява автоматизация на движението на данни между системите. Може да се използва за създаване на потоци от данни, които четат или пишат както на HDFS, така и на PostgreSQL. NiFi процесорите за HDFS и SQL бази данни могат да бъдат конфигурирани за работа с PostgreSQL.

Предимства:

- Потребителският интерфейс на NiFi осигурява лесна конфигурация и мониторинг на потоците от данни.
- Той предлага произход на данните, което е ценно за проследяване и отстраняване на грешки при проблеми с потока от данни.
- Той е проектиран да обработва данни от всички форми, размери и обеми.

Недостатъци:

- Като графичен инструмент, той може да не е толкова ефективен за сложна логика за трансформация на данни, колкото инструментите, базирани на код.
- Кривата на обучение за настройване на разширени конфигурации и оптимизации може да бъде стръмна.
- Възможно е да има затруднения в производителността за някои процесори, особено при голямо натоварване.

Пример:

NiFi поток от данни за преместване на данни от PostgreSQL към HDFS може да бъде настроен по следния начин:

GetJDBC процесор: Конфигуриран да се свърже с базата данни PostgreSQL и да изпълни SQL заявка за извличане на данните.

PutHDFS процесор: Конфигуриран да записва извлечените данни в HDFS.

Процесорите са свързани с връзка за данни и всеки процесор е конфигуриран с необходимите идентификационни данни и свойства на връзката. NiFi се грижи за останалото, включително повторен опит за неуспешни операции и буфериране на данни, ако е необходимо.

5. PostgreSQL опаковки за чужди данни (Foreign Data Wrappers - FDW)

Функцията за обвивка на чужди данни в PostgreSQL му позволява да действа като SQL-базирана система за федерирана база данни. HDFS може да бъде достъпен чрез използването на FDW, който се свързва с външни източници на данни. Тази интеграция позволява на PostgreSQL да търси директно данни, съхранявани в HDFS.

Предимства:

- FDW позволява данните да бъдат достъпни в различни системи, използвайки SQL възможностите на PostgreSQL.
- Той се интегрира безпроблемно в съществуващите PostgreSQL инсталации.
- Това може да бъде решение с ниски режийни разходи за комбиниране на SQL заявки за отдалечени източници на данни с локална обработка.

Недостатъци:

- Производителността често е ограничена от възможностите на FDW и латентността на мрежата между PostgreSQL и източника на данни.
- Тя може да бъде сложна за настройка и за изпълнение.
- Операциите за запис могат да бъдат ограничени или да не се поддържат в зависимост от конкретната реализация на FDW.

Пример:

За достъп до HDFS данни от PostgreSQL с помощта на FDW:

Инсталира се и се настройва Hadoop_FDW разширение в PostgreSQL.

Създава се чужд сървър, който се свързва с HDFS.

Съпоставят се външни таблици в HDFS към чужди таблици в PostgreSQL.

```
CREATE EXTENSION hadoop_fdw;

CREATE SERVER hdfs_server FOREIGN DATA WRAPPER hadoop_fdw
OPTIONS (host 'hdfs-hostname', port '8020');

CREATE FOREIGN TABLE hdfs_table (...) SERVER hdfs_server OPTIONS
(filename '/path/to/hdfs/file');

SELECT * FROM hdfs_table;
```

6. Конектори за бази данни и JDBC/ODBC

Различни компоненти на Hadoop, като Apache Hive или Apache Drill, имат достъп до релационни бази данни, използвайки JDBC (Java Database Connectivity) или ODBC (Open Database Connectivity). Тези компоненти могат да правят заявки към PostgreSQL бази данни директно и да съхраняват резултатите в HDFS.

Предимства:

- Осигуряват стандартизиран начин за свързване с PostgreSQL от различни приложения и услуги.
- JDBC / ODBC се поддържат от много инструменти, включително Hadoop екосистемни инструменти като Hive и Spark.
- Те обикновено са лесни за настройка и използване със съществуващи драйвери за бази данни.

Недостатъци:

- Производителността може да пострада поради режийните разходи на моста JDBC / ODBC, особено при големи обеми от данни.
- Те може да не поддържат всички функции на директна връзка с база данни.
- Могат да възникнат проблеми със съвместимостта на драйверите и е необходимо поддържането на актуални драйвери.

Пример:

За да преместите данни от HDFS в PostgreSQL с помощта на Hive с JDBC:

Настройва се таблица на Hive, която съпоставя данните в HDFS.

Използва се драйвера Hive JDBC за свързване с Hive от Java приложение.

Изпълнява се командата SQL INSERT INTO за преместване на данни от таблицата Hive в PostgreSQL.

```
Connection hiveConnection =
DriverManager.getConnection("jdbc:hive2://hive-
host:10000/default", "user", "password");

Statement stmt = hiveConnection.createStatement();

stmt.executeUpdate("INSERT INTO postgresql_table SELECT * FROM
hdfs_table");
```

7. ETL (Извличане, трансформиране, зареждане) инструменти

Традиционните ETL инструменти като Talend, Informatica и Pentaho могат да се използват за интегриране на данни между HDFS и PostgreSQL. Тези инструменти осигуряват графичен интерфейс за проектиране на тръбопроводи за данни и могат да се справят със сложни трансформации и пакетна обработка.

Предимства:

- ETL инструментите често идват с широк спектър от вградени конектори за различни източници на данни и цели, включително HDFS и PostgreSQL.
- Те осигуряват визуален интерфейс за проектиране на работни потоци от данни, което може да бъде по-лесно за потребителите, които не са запознати с програмирането.
- Усъвършенстваните ETL инструменти предлагат стабилни възможности за трансформация на данни, като се справят със сложни задачи за манипулиране на данни.

Недостатъци:

- Търговските ETL инструменти могат да бъдат скъпи поради разходите за лицензиране.
- Те могат да бъдат прекалено много за прости задачи за прехвърляне на данни, добавяйки ненужна сложност и режийни разходи.
- ETL процесите могат да бъдат ресурсоемки, особено когато се занимават с големи масиви от данни.

Пример:

Типичен ETL процес, използващ инструмент като Talend или Informatica, би включвал:

Създаване на задание или работен поток в ETL инструмента.

Конфигуриране на компонент източник за свързване и извличане на данни от HDFS.

По желание добавяне на компоненти за трансформация за обработка на данните.

Конфигуриране на компонент местоназначение за зареждане на данните в PostgreSQL.

Интерфейсът на ETL инструмента ще води потребителя през настройването на всеки компонент и картографирането на полета с данни между източника и местоназначението.

8. Промяна на снемането на данни (CDC)

Инструментите на CDC могат да улавят промените, направени на ниво база данни в реално време и да прилагат тези промени към друга система. Инструменти като Debezium могат да уловят промените на ниво ред в PostgreSQL и да предават тези промени на HDFS, използвайки Kafka като междинно хранилище на данни.

Предимства:

- CDC позволява интеграция на данни в реално време, улавяйки промените, докато се случват.
- Той минимизира натоварването на базата данни източник, тъй като улавя само постепенните промени.
- CDC е подходящ за архитектури, задвижвани от събития и модели на микроуслуги.

Недостатъци:

- Настройването на CDC може да бъде сложно и изисква дълбоко разбиране на вътрешните елементи на базата данни на източника.
- Това може да изисква допълнителна инфраструктура, като Apache Kafka, за буфериране и разпространение на събития за промяна.
- Справянето с извънредните събития и осигуряването на съгласуваност на данните може да бъде предизвикателство.

Пример:

Настройката на CDC, използваща Debezium и Kafka, ще включва:

Конфигуриране на Debezium за наблюдение на PostgreSQL за промени.

Стриймингът променя събитията в тема на Кафка.

Консумиране на темата на Кафка и писане на промените в HDFS, евентуално с помощта на Kafka-HDFS конектор или персонализирана потребителска работа.

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Arrays.asList("postgres_changes"));

for (ConsumerRecord<String, String> record : consumer.poll(Duration.ofMillis(100))) {

    writeToHdfs(record.key(), record.value());

}
```

9. Виртуализация на данни

Платформите за виртуализация на данни могат да осигурят интеграция в реално време между различни източници на данни, без физически да преместват данни. Те могат да

създадат виртуален слой, който обединява данни от HDFS и PostgreSQL, позволявайки на потребителите да търсят данни в тези източници, сякаш са едно цяло.

Предимства:

- Той осигурява унифициран слой за заявки в различни източници на данни, опростявайки достъпа и анализа.
- Избягва дублирането на данни и намалява необходимостта от движение на данни.
- Може да поддържа случаи на използване в реално време, като предлага почти мигновен достъп до основните източници на данни.

Недостатъци:

- Може да не е подходящ за тежки работни натоварвания поради потенциални въздействия върху производителността.
- Сложността на изпълнението и поддръжката може да бъде висока.
- Инструментите за виртуализация на търговски данни могат да дойдат със значителни разходи за лицензиране.

Пример:

Използвайки инструмент за виртуализация на данни като Denodo, човек би:

Свържете инструмента както към PostgreSQL, така и към HDFS като източници на данни.

Дефиниране на виртуални модели на данни, които представляват структурата от данни, изисквана от крайните потребители или приложения.

Запитване тези виртуални модели, които от своя страна запитване основните източници на данни в реално време.

10. HDFS клиентски библиотеки

Клиентските библиотеки за HDFS, като libhdfs за C или PyArrow за Python, могат да се използват за писане на персонализирани приложения, които взаимодействат както с HDFS, така и с PostgreSQL. Тези приложения могат програмно да преместват данни между двете системи.

Предимства:

- Осигурете програмен контрол върху операциите с данни, предлагайки гъвкавост за внедряване на персонализирана логика.

- Активиране на интегрирането на HDFS операции с данни в съществуващи приложения.
- Полезно за създаване на персонализирани решения, които отговарят на специфични изисквания.

Недостатъци:

- Разработването на персонализирани приложения изисква значителни усилия за кодиране и поддръжка.
- Оптимизацията на производителността може да бъде сложна и изисква дълбоко разбиране както на HDFS, така и на PostgreSQL.
- Потенциал за бъгове и проблеми, дължащи се на персонализиран код, който може да не е толкова щателно тестван, колкото стандартните инструменти.

Пример:

Скрипт на Python, използващ psycopg2 за PostgreSQL и PyArrow за HDFS, може да изглежда така:

```
import psycopg2

import pyarrow as pa

import pyarrow.hdfs as hdfs


conn = psycopg2.connect(database="dbname", user="user", password="password", host="hostname", port="port")

cursor = conn.cursor()


hdfs_connect = hdfs.connect(host='hdfs-hostname', port=8020)


cursor.execute("SELECT * FROM some_table")

rows = cursor.fetchall()


with hdfs_connect.open('/path/to/hdfs/file', 'wb') as file:

    file.write(rows)
```

Всеки от тези подходи има своите предимства и недостатъци по отношение на сложността, производителността, мащабируемостта и възможностите за обработка в реално време. Изборът на метод за интеграция до голяма степен ще зависи от конкретния случай на използване, обема и скоростта на данните и желаното ниво на свързване между системите.

Плюсовете и минусите на всеки подход за интегриране на HDFS с PostgreSQL база данни са обобщени в следната таблица:

#	**Подход**	**Предимства**	**Недостатъци**
1	Партиден (batch) трансфер на данни с помощта на Sqoop	Ефективен за големи партиди; паралелна обработка; се интегрира с екосистемата на Hadoop.	Не е подходящ за обработка в реално време; сложна настройка; режийни разходи на MapReduce.
2	Стрийминг на данни с Apache Flume	Добър за стрийминг на данни; Адаптивни; може да обработва потоци от данни с висока пропускателна способност.	Изисква персонализирана мивка за PostgreSQL; ограничена транзакционна подкрепа; стръмна крива на обучение.
3	MapReduce / Spark jobs	Много адаптивни; може да извършва сложна обработка; Мащабируеми.	Изисква значителни усилия за развитие; не е естествено интегрирана с PostgreSQL.
4	Apache NiFi	Визуално управление на потока от данни; поддържа широки източници на данни и дестинации; Добре е за потеклото на данните.	Може да бъде сложен за конфигуриране; потенциални затруднения в производителността.
5	PostgreSQL опаковки за чужди данни (FDW)	Разрешава SQL заявки за HDFS данни; се интегрира с оптимизатора на заявки на PostgreSQL.	Ограничено от изпълнението на FDW; допълнителни режийни разходи за настройка и поддръжка.
6	Конектори за бази данни и JDBC/ODBC	Стандартен интерфейс за достъп до данни; широко подкрепени от инструменти за данни; лесна интеграция с BI инструменти.	Може да въведе режийни разходи за изпълнение; ограничена от ефективността на JDBC / ODBC драйвера.
7	ETL инструменти	Графичен интерфейс за проектиране на тръбопроводи; поддържа сложни трансформации; пакетна обработка.	Може да бъде ресурсоемко; разходи за лицензиране на търговски инструменти; потенциална крива на обучение.
8	Промяна на снемането на данни (CDC)	Синхронизация на данни в реално време; минимално въздействие върху системата източник; Подходящ за архитектури, задвижвани от събития.	Комплексна настройка; изисква допълнителна инфраструктура като Kafka; потенциални предизвикателства, свързани с последователността на данните.
9	Виртуализация на данни	Интеграция в реално време; избягва дублирането на данни; унифицирани заявки между източници на данни.	Може да бъде сложен за изпълнение; потенциално въздействие върху ефективността; разходи за лицензиране на търговски платформи.

10	HDFS клиентски библиотеки	Програмен контрол върху достъпа до данни; поддържа сложна логика; Подходящ за персонализирани приложения.	Режийни разходи за развитие; изисква поддръжка на персонализиран код; потенциални проблеми с производителността.

Литература

1. WebHDFS FileSystem APIs; 2022; https://learn.microsoft.com/en-us/rest/api/datalakestore/webhdfs-filesystem-apis
2. Cloudera; Apache Nifi; https://www.cloudera.com/products/open-source/apache-hadoop/apache-nifi.html
3. Abhinav Agarwal; Use NiFi to extract and parse data from HTTP endpoints; 2023; https://www.projectpro.io/recipes/use-nifi-extract-and-parse-data-from-http-endpoints-and-store-data-persistent-storage
4. Flume 1.11.0 User Guide; https://flume.apache.org/releases/content/1.11.0/FlumeUserGuide.html
5. David Taylor; Apache Flume Tutorial: What is, Architecture & Hadoop Example; https://www.guru99.com/create-your-first-flume-program.html
6. Prateek Majumder; Apache Sqoop: Features, Architecture and Operations; 2023; https://www.analyticsvidhya.com/blog/2022/09/apache-sqoop-features-architecture-and-operations/
7. Foreign Data Wrappers; https://wiki.postgresql.org/wiki/Foreign_data_wrappers
8. PostgreSQL Documentation – Create foreign data wrapper; https://www.postgresql.org/docs/current/sql-createforeigndatawrapper.html