Санкт-Петербургский государственный университет
Математико-Механический факультет
Кафедра прикладной кибернетики

Работа допущена к защите
зав. кафедрой

_____

«____» _____ 2021 г.

Отчёт по производственной практике

Системы загрузки больших данных

Направление: 010400 (01.03.02) – Прикладная математика и информатика

Выполнил студент гр. 323 _____ Мамаев Владислав Викторович

Научный руководитель,
к. ф.-м. н. _____ Благов Михаил Валерьевич

Санкт-Петербург – 2021

Отзыв научного руководителя

Тема работы является актуальной. При выполнении работы студент проявил самостоятельность. Работа выполнена на хорошем уровне, отчёт составлен грамотно и достаточно полно отражает результаты работы студента.

Производственная практика выполнена полностью и может быть зачтена.

В ходе работы студент приобрел следующие компетенции: ПКП-1, ПКП3, УКБ-5.

Оценка А

к. ф.-м. н. Благов Михаил Валерьевич

Оглавление

1. Введение

Базы данных используются повсеместно, при этом системы, для которых они предназначены, можно разделить на 2 класса:[1]

- Online Transaction Processing (OLTP) – системы обработки транзакций в реальном времени. Такие системы используются для операционной деятельнос предприятий.
- Online Analytical Processing (OLAP) – системы аналитической обработки в реальном времени. К такими системам относятся системы поддержки принятия решений, инструменты business intelligence, системы анализа данных.

OLAP системы содержат информацию из OTLP систем, при обычно OLAP системы обычно функционирует независимо от OLTP систем по следующим причинам:[2]

- Реляционная схема данных, обычно используемая в OLTP системах, не эффективна для OLAP нагрузки.
- OLAP нагрузка на OLTP систему может вызвать проблемы с производительнос транзакций.
- С увеличением размера хранимых данных, возникают ограничения на используемые технологии и существенно увеличивается стоимость хранения данных в системах не предназначенных только для OLAP.
- Необходимость доступа к данным из разных OLAP систем и возможности ограничения доступа к данным

Традиционно, данные попадают в OLAP систему из OLTP системы при помощи Extraction-Transformation-Loading (ETL) программ, запускаемых с некоторо

периодичностью. ETL процессы могут занимать достаточно много времени, и это создает задержку появления данных в OLAP системе.

Для того чтобы уменьшить задержку можно вместо периодичных ETL процессов использовать потоковую обработку. Подобная архитектура интеграции данных описана в [3] [4]. Каждое изменение данных в OLTP системе записывается в очередь сообщений, а некоторая программа непрерывно читает сообщения и обновляет данные в аналитическом хранилище. Использование непрерывных обновлений уменьшает задержку, но требует специальных инструментов для обеспечения целостности данных — традиционно используемые хранилища для большого объема данных, такие как Hadoop, работают по принципу write-once и не поддерживают обновлений.

1.1. Постановка задач

Цель работы — познакомиться с инструментами для работы с большими данными на примере задачи интеграции OLTP и OLAP хранилищ. Задачи:

- Сконфигурировать кластер Hadoop и распределённую файловую систему HDFS.
- Сконфигурировать кластер Spark для распределённой обработки данных.
- Сконфигурировать кластер Kafka для распределённой очереди сообщений.
- Реализовать алгоритм записи csv файла в таблицы для больших данных, используя Apache Hudi и Delta Lake.
- Реализовать алгоритм потоковой записи из Kafka топика в таблицы для больших данных, используя Apache Hudi и Delta Lake.
- Реализовать выполнение некоторых запросов к таблицам для больших данных.

2. Основная часть

2.1. Описание используемых технологий

Apache Spark — это среда выполнения для обработки больших данных. Он предоставляет программные интерфейсы на Java, Scala, Python и R, а также оптимизированный движок, поддерживающий выполнение задач с произвольным графом исполнения. Он также поддерживает богатый набор инструментов более высокого уровня, включая Spark SQL для SQL и обработки структурированных данных, MLlib для машинного обучения, GraphX для обработки графов и структурир потоковую передачу для инкрементных вычислений и потоковой обработки.[5]

Delta Lake — это проект с открытым исходным кодом, который позволяет построить архитектуру Lakehouse на основе озер данных. Delta Lake обеспечивает ACID транзакции, масштабируемую обработку метаданных и унифицирует потокову и пакетную обработку данных поверх существующих озёр данных, таких как S3, ADLS, GCS и HDFS.[6]

Apache Hudi — это среда управления данными с открытым исходным кодом, используемая для упрощения потоковой обработки данных и разработки конвейеров данных.[7]

Apache Kafka — это платформа для потоковой передачи событий имеющая три ключевые возможности:[8]

- Публиковать (писать) и подписываться (читать) потоки сообщений, включая непрерывный импорт или экспорт данных из других систем.
- Надёжно хранить потоки сообщений.
- Обрабатывать потоки сообщений по мере их возникновения или ретроспективно

2.2. Реализация

На листингах 1 2 3 4 5 6 приведены ключевые фрагменты реализации поставленных задач. Полная версия доступна по ссылке https://github.com/Vlad-commits/cw3

```
SparkSession spark = SparkSession.builder()
        .appName("Hudi ETL")
        .config("spark.master", "local")
        .config("spark.executor.memory", "8g")
        .config("spark.driver.memory", "4g")
        .config("spark.rdd.compress", "true")
        .config("spark.driver.maxResultSize", "2g")
        .config("spark.serializer",
           "org.apache.spark.serializer.KryoSerializer")
        .config("spark.kryoserializer.buffer.max", "512m")
        .getOrCreate();

    Dataset<Row> data = spark.read()
        .option("header", true)
        .csv(CSV_PATH);

    //Write
    data.write()
        .format("hudi")
        .option("hoodie.insert.shuffle.parallelism", "1")
        .option("hoodie.upsert.shuffle.parallelism", "1")
        .option("hoodie.clean.automatic", "false")
        .option("hoodie.bulkinsert.sort.mode",
           BulkInsertSortMode.NONE.toString())
        .option(DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY(),
           "timestamp")
        .option(DataSourceWriteOptions.RECORDKEY_FIELD_OPT_KEY(), "uuid")
        .option(DataSourceWriteOptions.PARTITIONPATH_FIELD_OPT_KEY(), "")
```

```
    .option(HoodieWriteConfig.TABLE_NAME, TABLE_NAME)
    .mode(Append)
    .save(HUDI_PATH);

//Read
Dataset<Row> snapshotDf = spark.read().
    format("hudi").
    load(HUDI_PATH + "/*/");
snapshotDf.show();
```

Листинг 1. Запись из CSV в Hudi таблицу

```
SparkSession spark = SparkSession.builder()
    .appName("Hudi ETL")
    .config("spark.master", "local")
    .config("spark.executor.memory", "8g")
    .config("spark.driver.memory", "4g")
    .config("spark.rdd.compress", "true")
    .config("spark.driver.maxResultSize", "2g")
    .config("spark.serializer",
       "org.apache.spark.serializer.KryoSerializer")
    .config("spark.kryoserializer.buffer.max", "512m")
    .getOrCreate();

StreamingQuery query = spark
    .readStream()
    .format("kafka")
    .option("kafka.bootstrap.servers", "localhost:9092")
    .option("subscribe", "verytesttopic123")
    .option("startingOffsets", "latest")
    .load()
    .selectExpr("CAST(value AS STRING)")

    .writeStream()
    .foreachBatch((v1, v2) -> {
       v1.persist();
       v1.write()
           .format("hudi")
           .option("hoodie.bulkinsert.sort.mode",
              BulkInsertSortMode.NONE.toString())
           .option("hoodie.insert.shuffle.parallelism", "1")
           .option("hoodie.upsert.shuffle.parallelism", "1")
           .option("hoodie.clean.automatic", "false")
           .option(DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY(),
              "timestamp")
           .option(DataSourceWriteOptions.RECORDKEY_FIELD_OPT_KEY(),
```

```
                "uuid")
            .option(DataSourceWriteOptions.PARTITIONPATH_FIELD_OPT_KEY(),
                "")
            .option(HoodieWriteConfig.TABLE_NAME, TABLE_NAME)
            .mode(Append)
            .save(HUDI_PATH);
        v1.unpersist();
    })
    .option("checkpointLocation", CHECKPOINT_PATH)
    .start();

query.processAllAvailable();
```

Листинг 2. Запись из Kafka в Hudi таблицу

```
SparkSession spark = SparkSession.builder()
    .appName("Hudi Query")
    .config("spark.master", "local")
    .config("spark.serializer",
       "org.apache.spark.serializer.KryoSerializer")
    .config("spark.kryoserializer.buffer.max", "256")
    .getOrCreate();


Dataset<Row> snapshotDf = spark.read().
    format("hudi").
    load(HUDI_PATH + "/*/");
snapshotDf.createOrReplaceTempView("temp_events");

spark.sql("select count(1) from temp_events ").show();
spark.sql("select sensorId , count(1) from temp_events group by
   sensorId").show();
spark.sql("select * from (select temp_events.* , row_number() over
   (partition by sensorId order by timestamp desc) as rnk from
   temp_events) where rnk=1").show();
```

Листинг 3. Выполнение запросов к Hudi таблице

```
SparkSession spark = SparkSession.builder()
    .appName("Delta Lake ETL")
    .config("spark.master", "local")
    .config("spark.sql.extensions",
       "io.delta.sql.DeltaSparkSessionExtension")
    .config("spark.sql.catalog.spark_catalog",
       "org.apache.spark.sql.delta.catalog.DeltaCatalog")
    .getOrCreate();

Dataset<Row> data = spark.read()
    .option("header", true)
    .csv(CSV_PATH);
data.show();



// Init table
DeltaTable deltaTable = DeltaTable.forPath(PATH);

// Upsert (merge) new data

deltaTable.as("oldData")
    .merge(
       data.as("newData"),
       "oldData.uuid = newData.uuid")
    .whenMatched()
    .update(
       new HashMap<String, Column>() {{
           put("uuid", functions.col("newData.uuid"));
           put("sensorId", functions.col("newData.sensorId"));
           put("eventDescription",
              functions.col("newData.eventDescription"));
           put("timestamp", functions.col("newData.timestamp"));
       }})
    .whenNotMatched()
```

```
    .insert(
        new HashMap<String, Column>() {{
            put("uuid", functions.col("newData.uuid"));
            put("sensorId", functions.col("newData.sensorId"));
            put("eventDescription",
                functions.col("newData.eventDescription"));
            put("timestamp", functions.col("newData.timestamp"));
        }})
    .execute();

deltaTable.toDF().show();


//Read
Dataset<Row> df = spark.read()
    .format("delta")
    .load(PATH);
df.show();
```

Листинг 4. Запись из CSV в Delta Lake таблицу

```
SparkSession spark = SparkSession.builder()
    .appName("Delta Lake ETL")
    .config("spark.master", "local")
    .config("spark.sql.extensions",
       "io.delta.sql.DeltaSparkSessionExtension")
    .config("spark.sql.catalog.spark_catalog",
       "org.apache.spark.sql.delta.catalog.DeltaCatalog")
    .getOrCreate();

DataStreamWriter<Row> writeStream = spark
    .readStream()
    .format("kafka")
    .option("kafka.bootstrap.servers", Utils.getKafkaHosts())
    .option("subscribe", Utils.getTopic())
    .option("startingOffsets", "latest")
    .load()
    .selectExpr("CAST(value AS STRING)")
    .writeStream();
DeltaTable deltaTable = DeltaTable.forPath(deltaTableName);

StreamingQuery query = writeStream
    .option("checkpointLocation", CHECKPOINT_PATH)
    .foreachBatch(((v1, v2) -> {
       v1.persist();
       deltaTable.as("oldData")
           .merge(
              v1.as("newData"),
              "oldData.uuid = newData.uuid")
           .whenMatched()
           .update(
              new HashMap<String, Column>() {{
                 put("uuid", functions.col("newData.uuid"));
                 put("sensorId", functions.col("newData.sensorId"));
                 put("eventDescription",
```

```
                        functions.col("newData.eventDescription"));
                    put("timestamp", functions.col("newData.timestamp"));
                }})
            .whenNotMatched()
            .insert(
                new HashMap<String, Column>() {{
                    put("uuid", functions.col("newData.uuid"));
                    put("sensorId", functions.col("newData.sensorId"));
                    put("eventDescription",
                        functions.col("newData.eventDescription"));
                    put("timestamp", functions.col("newData.timestamp"));
                }})
            .execute();
        v1.unpersist();
    }))
    .start();
query.processAllAvailable();
```

Листинг 5. Запись из Kafka в Delta Lake таблицу

```
SparkSession spark = SparkSession.builder()
    .appName("Delta Lake Query")
    .config("spark.master", "local")
    .config("spark.sql.extensions",
       "io.delta.sql.DeltaSparkSessionExtension")
    .config("spark.sql.catalog.spark_catalog",
       "org.apache.spark.sql.delta.catalog.DeltaCatalog")
    .getOrCreate();

Dataset<Row> df = spark.read()
    .format("delta")
    .load(PATH);
df.createOrReplaceTempView("temp_events");
spark.sql("select count(1) from temp_events ").show();
spark.sql("select sensorId , count(1) from temp_events group by
   sensorId").show();
spark.sql("select * from (select temp_events.* , row_number() over
   (partition by sensorId order by timestamp desc) as rnk from
   temp_events) where rnk=1").show();
```

Листинг 6. Выполнение запросов к Delta Lake таблице

3. Заключение

В рамках работы были успешно реализованы алгоритмы для записи, потоковой записи и чтения для таблиц больших данных на базе Apache Hudi и Delta Lake.

Список литературы

1. *Chaudhuri S.*, *Dayal U.* An overview of data warehousing and OLAP technology // ACM Sigmod record. — 1997. — Т. 26, № 1. — С. 65—74.
2. *Conn S. S.* OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis // Proceedings. IEEE SoutheastCon, 2005. — 2005. — С. 515—520. — DOI: 10.1109/SECON.2005.1423297.
3. *Tho M. N.*, *Tjoa A. M.* Zero-latency data warehousing for heterogeneous data sources and continuous data streams // 5th International Conference on Information Integrationand Web-based Applications Services. — 2003. — С. 55—64.
4. *Kakish K.*, *Kraft T. A.* ETL evolution for real-time data warehousing // Proceedings of the Conference on Information Systems Applied Research ISSN. Т. 2167. — 2012. — С. 1508.
5. Оффициальный сайт Spark. — URL: https://spark.apache.org/.
6. Оффициальный сайт Delta Lake. — URL: https://delta.io/.
7. Оффициальный сайт Apache Hudi. — URL: https://hudi.apache.org/.
8. Оффициальный сайт Apache Kafka. — URL: https://kafka.apache.org/.