УДК 004.65
DOI https://doi.org/10.24144/2616-7700.2024.44(1).66-82

М. І. Глебена[1], А. В. Макарович[2]

[1] ДВНЗ «Ужгородський національний університет»,
завідувач кафедри системного аналізу та теорії оптимізації,
кандидат фізико-математичних наук
myroslava.hlebena@uzhnu.edu.ua
ORCID: https://orcid.org/0000-0003-1100-515X

[2] Київський національний університет ім. Т. Шевченка,
здобувач другого (магістерського) рівня вищої освіти, ОНП "Системи і методи прийняття рішень"
adalbert.makarovych@gmail.com
ORCID: https://orcid.org/0009-0000-2352-9933

SINGLESTOREDB КОНЕКТОР ДЛЯ APACHE BEAM

В статті досліджено процес розробки SingleStoreDB конектора для Apache Beam та складнощі, які виникли під час цього процесу. Для реалізації конектора використано мову програмування Java. Для компіляції та менеджменту залежностей проекту використано систему автоматичного збирання Gradle. Для забезпечення неперервного тестування конектора використано інструмент для неперервної інтеграції Jenkins. Задля комунікації із базою даних використано бібліотеку SingleStore JDBC Driver. Об'єктом дослідження обрано процес обміну даних між Apache Beam та SingleStoreDB. Предметом дослідження є конектор, що дозволяє ефективно обмінюватись даними між Apache Beam та SingleStoreDB. Для розробки конектора проаналізовані вимоги до нього. Продемонстровано, що основними вимогами є можливість паралельного запису даних, можливість паралельного читання даних за допомогою розподілених властивостей SingleStoreDB, можливість виконати запити читання даних непаралельно для специфічних запитів. Досліджено можливість використання різних способів паралельного читання та запису даних. Згідно досліджень найоптимальнішим способом паралельного читання є розбиття запиту на декілька незалежних використовуючи певну колонку. Найоптимальнішим способом запису даних є використання запиту LOAD DATA. На відміну від INSERT запиту надсилає дані як окремий потік байтів, а не як частину запиту. За рахунок цього, дані можна відправляти набагато більшими групами. Протестовано конектор та налаштовано систему неперервної інтеграції. Розроблено документацію та інтегровано конектор в GitHub репозиторій Apache Beam. Розроблений конектор може бути використаний для ETL процесів під час яких Apache Beam відповідає за обробку та збереження даних в SingleStoreDB, що надає зручніший спосіб конфігурації та кращу продуктивність в порівнянні із JdbcIO конектором. Результати досліджень можуть бути використані для створення конекторів між іншими технологіями та СКБД SingleStoreDB.

Ключові слова: база даних, конектор, розподілені системи, ETL, Apache Beam, SingleStore, OLAP, OLTP.

1. Вступ. Apache Beam вважається наступним кроком в еволюції розподілених систем обробки даних [25]. Існуючий JdbcIO конектор дозволяє з'єднання із більшістю реляційних баз даних, однак, він не враховує їхні особливості. SingleStoreDB є розподіленою системою керування базами даних. Ця СКБД підтримує як OLAP так і OLTP способи використання. За рахунок цього, вона набула популярності для аналізу великих обсягів даних. Архітектура SingleStoreDB передбачає можливості для паралелізації запитів запису та читання. Ці можливості не враховані в JdbcIO конекторі.

Apache Beam містить більше 50 конекторів, які інтегровані в систему і підтримуються розробниками APACHE SOFTWARE FOUNDATION. Крім цього, існує 10 конекторів, які не вбудовані в саму систему і підтримуються третіми сторонами. Незважаючи на таку велику кількість інтеграцій, існує величезна кількість технологій, з якими Apache Beam ще не інтегрований.

Якщо розглядати використання Apache Beam із реляційними базами даних, які використовують мову SQL, то найзагальнішим способом інтеграції таких технологій є використання JDBC драйверів. Apache Beam включає в себе JdbcIO конектор. За допомогою цієї бібліотеки, система може під'єднуватись до будь-якої реляційної бази даних, яка має свій JDBC драйвер. Таке рішення є найпростішим способом розширити кількість підтримуваних технологій.

Таким чином, на даний момент, можливо читати та записувати дані в базу даних SingleStoreDB використовуючи JdbcIO. Але цей спосіб не враховує особливості бази даних SingleStoreDB. Через це конектор має не найкращу швидкодію та надає не найкращий користувацький досвід. Це зумовлює необхідність розробки конектора, який враховував би особливості SingleStoreDB та надавав якіснішу практику використання для користувачів.

2. Основний результат. *Об'єктом дослідження* є процес обміну даних між Apache Beam та SingleStoreDB.

Предметом дослідження є конектор, що дозволяє ефективно обмінюватись даними між Apache Beam та SingleStoreDB.

Мета роботи полягає у реалізації Apache Beam конектора для SingleStoreDB, який би враховував можливості та особливості цієї СКБД.

Для досягнення поставленої мети виокремленні наступні завдання:

- сформулювати вимоги до конектора, необхідні для задоволення потреб користувачів;
- реалізувати та протестувати конектор;
- інтегрувати конектор в GitHub репозиторій Apache Beam.

Матеріал і методи досліджень. Для реалізації конектора використано мову програмування Java. Для компіляції та менеджменту залежностей проекту використано систему автоматичного збирання Gradle. Для того, щоб забезпечити неперервне тестування конектора використано інструмент для неперервної інтеграції Jenkins. Для комунікації із базою даних використано бібліотеку SingleStore JDBC Driver.

Аналіз останніх досліджень та публікацій. В 2017 році APACHE SOFTWARE FOUNDATION оголосила Apache Beam проектом найвищого рівня [22]. Ця технологія базується на моделі програмування MapReduce опублікованій в 2004 році розробниками компанії Google [12]. В цій науковій роботі продемонстровано, що за допомогою цієї моделі можна виразити багато прикладних завдань. Програми, написані в цьому функціональному стилі, автоматично розпаралелюються та виконуються на великому кластері стандартних машин. Це дозволяє програмістам без жодного досвіду роботи з паралельними та розподіленими системами, легко використовувати ресурси таких систем.

Apache Beam може бути використаний для ETL завдань та інтеграції даних. Такі завдання корисні для переміщення великих обсягів даних між джерелами даних, трансформації їх до необхідного формату або завантаження даних в нові системи [3].

Apache Beam розроблений таким чином, щоб користувачі могли реалізувати новий I/O конектор. Для цього завдання розроблені спеціальні інтерфейси та набір інструкцій та рекомендацій [8, 9, 10, 15].

SingleStoreDB це розподілена реляційна СКБД [11]. За рахунок розподіленості запити читання та завантаження даних можуть бути розпаралелені. SingleStoreDB надає можливості швидкого завантаження даних за допомогою запитів LOAD DATA [18], а також можливість паралельного читання даних [19]. Ці особливості СКБД були вдало використані для створення конектора із Apache Spark, а отже перспективним є дослідження використання їх при інтеграції із Apache Beam.

Результати дослідження та їх обговорення. Створення якісного конектора вимагає детального аналізу особливостей систем, які він має поєднувати. Також важливо виділити ситуації, в яких користувачі будуть використовувати ці системи разом. Після проведення цього аналізу, можна виділити найважливіший функціонал та сформулювати вимоги до конектора.

SingleStoreDB це розподілена реляційна СКБД [11]. Ця СКБД представляє собою кластер із декількома вузлами. Кожен вузол є процесом запущеним на конкретній машині. Всі вузли поділяються на два типи:

- Агрегатори — відповідають за передачу запитів до листків і надсилання результату виконання запитів до клієнта
- Листки — вузли, що зберігають частину даних.

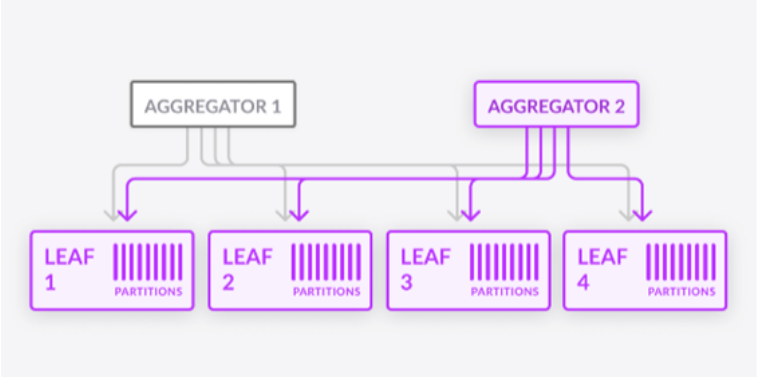

Рис. 1. Архітектура SingleStoreDB кластера [11].

Для оптимізації швидкодії, SingleStoreDB автоматично розбиває дані на розділи. Кожен листок може зберігати декілька розділів [6].

За рахунок такої архітектури, SingleStoreDB може паралельно обробляти дані на декількох вузлах незалежно. Також, це дозволяє користувачам горизонтально розширювати їхній кластер у випадку, якщо їм потрібно більше обчислювальних ресурсів. Користувачі здатні надсилати запити до різних агрегаторів, щоб не бути обмеженими швидкістю мережі однієї машини.

Ще однією особливістю SingleStoreDB є універсальне сховище, яке є еволюцією columnstore таблиць. На відміну від стандартної реалізації columnstore таблиць, реалізація SingleStoreDB підтримує транзакційні процеси, для яких зазвичай доводиться використовувати rowstore таблиці [24]. Ця особливість дозволяє одночасно завантажувати та оновлювати дані в таблицях, та проводити

на них аналітичні запити, що містять агрегації. За рахунок цієї особливості SingleStoreDB і отримала свою назву.

Кластер SingleStoreDB може бути запущений у двох варіантах:

- Cloud
- Self-managed

При використанні Cloud, кластер створюється і конфігурується автоматично. Клієнт отримує єдину адресу, яку він може використовувати для надсилання запитів до бази даних. Всі запити розподіляються між агрегаторами, балансуючи навантаження на кластер. Крім того, такий спосіб розгортання надає можливість використання bottomless. Це особливість бази даних, яка передбачає, що самі дані будуть відвантажуватись на S3. За рахунок, цього ресурси необхідні для виконання обчислень та ресурси необхідні для збереження даних стають незалежними. Користувач може зберігати терабайти даних на маленькому кластері і платити за сховище незалежно ресурсів необхідних для виконання операцій над ними [14].

Self-managed розгортання надає клієнту можливість розгорнути кластер на своїх серверах та встановити набагато більше особливих конфігурацій.

Apache Beam є уніфікованою моделлю програмування та множиною бібліотек для створення та виконання конвеєрів для обробки даних [3]. В 2004 році компанія Google опублікувала наукову роботу про модель програмування MapReduce [12]. Ця модель базується на двох операціях Map і Reduce. В роботі показано, що багато реальних задач можна виразити через цю модель. При цьому, програми написані на основі такої моделі можна паралелізувати і виконувати на великих кластерах. Пізніше на основі цієї моделі було створено багато реалізацій таких як Hadoop, Flume, Spark, Flink. Наступним кроком в еволюції цих моделей було створення Apache Beam. Основними особливостями Apache Beam є:

- уніфікована мова програмування для пакетної та потокової обробки даних;
- велика кількість підтримуваних джерел для запису та збереження даних;
- можливість виконання обчислень на різних системах (наприклад, на тих самих Flink, Spark);
- наявність SDK для різних мов програмування (Go, Java, Python, Typescript).

Розглянемо кожну з цих особливостей окремо.

В старих фреймворках для паралельної обробки даних таких як Hadoop, Flink, Spark реалізовано різний API для пакетної та потокової обробки даних [25]. Для прикладу, в Apache Spark для пакетної обробки використовувались RDD та Dataframe, а для потокової — Datastreams.

На Рис. 3. можна побачити приклади коду пакетної та потокової обробки даних в Spark.

На відміну, від цих фреймворків, Apache Beam надає єдиний API для обробки пакетових та потокових даних.

Наступною особливістю Apache Beam є портативність. Ця модель не виконує операції сама по собі, а замість цього делегує виконання іншим системам. Це робить компанії незалежними від технологій, які вони використовують. Якщо компанія використовувала Spark кластер для виконання обчислень, але з часом помітила, що Flink має кращу швидкодію, то вони можуть легко мігрувати на

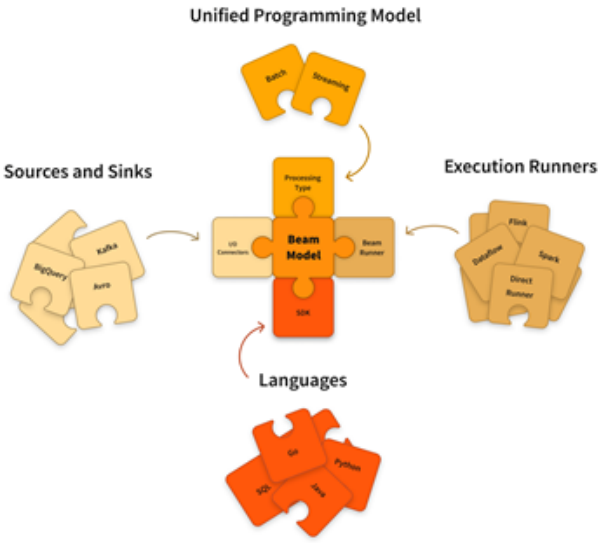

Рис. 2. Особливості Apache Beam [3].

Spark batch example

```
val sc = new SparkContext(conf)
val lines = sc.textFile("...")
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word -> (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

wordCounts.show()
```

Spark streaming example

```
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word -> (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

wordCounts.print()
```

Рис. 3. Приклад потокової та пакетової обробки даних в Spark.

```
pipeline = beam.pipleline()

data = pipeline | "ReadData" >> beam.io.ReadFromText("data.txt")
processed_data = data | "ProcessData" >> beam.Map(lambda x: x.upper())
processed_data | "WriteData" >> beam.io.WriteToText("processed_data.txt")

pipeline.run()
```

Рис. 4. Приклад обробки пакетових і потокових даних в Apache Beam.

нову технологію. Для цього не доведеться готувати всіх інженерів і переписувати існуючий код.

Ще одна особливість це підтримка багатьох мов програмування. Подібний

функціонал реалізований і в інших системах, але в Apache Beam є можливість використання функціоналу написаного на різних мовах всередині одного конвеєра. Це дозволяє розробникам використовувати улюблену мову програмування і інтегрувати частини коду всередині однієї програми.

Останньою особливістю є підтримка багатьох джерел для запису та збереження даних. Розробники Apache Beam надають велику кількість рекомендацій для створення конекторів до їхньої технології [15].

Для того, щоб підтримувати всі згадані вище особливості модель програмування має бути максимально простою. Кожна зайва абстракція буде значно ускладнювати процес підтримки уніфікованості та портативності моделі.

Основними абстракціями Beam є [5]:

- Pipeline (конвеєр) — напрямлений ациклічний граф даних та операцій над ними
- PCollection — невпорядкована множина елементів, яка відповідає вершинам графу. Зазвичай дані в цій множині розподілені.
- PTransform — перетворення над PCollection, що відповідають ребрам графу

Зазвичай Beam конектор складається з декількох PTransform (для читання та для запису даних). PTransform для читання або запису даних представляє собою клас, що містить функцію читання/запису. Ця функція буде паралельно викликатись для того, щоб прочитати або записати частину даних.

Враховуючи зазначені вище особливості можна припустити, що найбільш поширеною ситуацією, в якій користувач захоче інтегрувати ці дві технології буде випадок використання Apache Beam для ETL процесу, а SingleStoreDB для збереження даних та проведення їхнього аналізу. Apache Beam надає просту модель, яку легко вивчити та інтегрувати для того, щоб завантажувати дані з різних джерел, трансформувати їх та зберігати в базі даних. SingleStoreDB в свою чергу надає можливість швидко зберігати великі множини даних та обробляти їх після. В такий спосіб можна реалізувати аналітику в реальному часі над даними зібраними із різних джерел.

Для того, щоб максимально використати можливості обох розподілених систем — конектор має мати можливість паралельного запису та читання даних. Існуючий JdbcIO конектор має такі можливості. Проте, для запису даних він використовує INSERT запити [17]. SingleStore дозволяє завантажувати дані за допомогою LOAD DATA запитів [18]. Другий спосіб є набагато оптимальнішим, адже при ньому всі рядки таблиці відправляються окремим файлом і не мають бути закодовані всередині самого запиту.

Щодо паралельного читання, то JdbcIO конектор розбиває SELECT запит на декілька SELECT запитів із фільтром по певній колонці. Наприклад, замість запиту

SELECT a, b FROM t

конектор може надіслати наступні запити до бази даних

SELECT a, b FROM t WHERE $0 < a < 100$

SELECT a, b FROM t WHERE $100 < a < 200$

SELECT a, b FROM t WHERE $200 < a < 300$

Такий спосіб паралелізації запитів зможе працювати з більшістю реляційних баз даних. З іншого боку він має ряд недоліків. Користувач має вибрати колонку для розбиття. Вказати верхню та нижню межу (інакше конектор буде

запускати додаткові запити для визначення цих параметрів). Також важливо пересвідчитись, що вибрана колонка має індекс і що дані в ній рівномірно розподілені.

SingleStore є розподіленою базою даних і вже містить в собі функціонал для розбиття множини рядків на підмножини для виконання обчислень. Найкращим способом розпаралелити читання результату SELECT запиту є RESULT TABLES [19]. Доцільним є проведення досліджень щодо використання цього функціоналу у новому конекторі для Apache Beam.

Деякі запити до бази даних не можуть бути розпаралелені (наприклад SHOW TABLES). Для них доцільно розробити можливість непаралельного читання.

Отже вимогами до конектора будуть:

- можливість паралельного запису даних за допомогою LOAD DATA
- можливість паралельного читання даних за допомогою розподілених властивостей SingleStoreDB
- можливість виконати запити читання даних непаралельно для специфічних запитів.

Найпопулярнішим способом з'єднання з базами даних, які підтримують SQL є використання JDBC API. Використовуючи JDBC API, застосунки написані на мові Java можуть виконувати SQL запити, діставати результати та поширювати назад до відповідних джерел даних. З часів появи в січні 1997 року JDBC API став широко поширеним і реалізованим. Компанія SingleStore розробила свій власний JDBC драйвер. Цей драйвер сумісний з версією стандарту 4.2 та ліцензований згідно GNU LGPL. Ця бібліотека є найкращим вибором для комунікації Javaзастосунків із SingleStore.

Перед розробкою конектора була проведена консультація з розробниками компанії Google (LukaszCwik, KennethKnowles, BrianHulette, ChamikaraJayalath, JohnCasey) та розроблена технічна документація [http://surl.li/haior] відповідно до зразка [http://surl.li/haiou].

Всі параметри необхідні для створення з'єднання із базою даних інкапсульовані в єдиному класі DataSourceConfiguration. Цей клас відповідальний за валідацію та встановлення всіх параметрів а також за створення з'єднання за допомогою JDBC драйвера.

Клас DataSourceConfigutration надає можливість встановлення певних параметрів.

Endpoint — ім'я хоста або IP адреса бази даних в наступному форматі host:[port].
Username — ім'я користувача бази даних (за замовчуванням — root).
Password — пароль для користувача (за замовчуванням — пуста стрічка).
Database — база даних, яка буде використана за умови, якщо іншої не вказано в запиті.
ConnectionPropertirs — список конфігурацій для JDBC драйвера.
Параметр вказується в наступному форматі "key1=value1;key2=value2;...; keyN=valueN".

Список всіх підтримуваних параметрів доступний за посиланням [23].
Параметр endpoint є обов'язковим.
Приклад коду для створення об'єкта DataSourceConfiguration.

```
SingleStoreIO.DataSourceConfiguration
    .create("myHost:3306")
```

```
.withDatabase("db")
.withConnectionProperties("connectTimeout=30000")
.withPassword("password")
.withUsername("admin");
```

Результати деяких запитів в SingleStore не можна прочитати паралельно. Прикладами таких запитів є “SHOW TABLES”, “SHOW DATABASES”.

Саме для таких запитів необхідно реалізувати PTransform, який би просто виконував запит та повертав його результат. При імплементації паралельного читання можна використовувати досвід отриманий при реалізації простішої версії.

Цей PTransform приймає на вхід спеціальну початкову множину (PBegin) і повертає множину типів визначених користувачем (PCollection<T>). Для того, щоб запустити процес читання використано PTransformParDo. Він створить єдиний екземпляр класу DoFn та запустить єдину функцію для виконання запиту та читання результату.

Налаштувати це перетворення можна за допомогою наступних параметрів:

- dataSourceConfiguration — об’єкт типу DataSourceConfiguration з усією інформацією необхідною для створення з’єднання
- query — SQL запит, що має бути виконаний
- table — таблиця з якої читати дані
- statementPreparator — об’єкт що реалізує інтерфейс StatementPreparator. Цей об’єкт має реалізовувати функцію setParameters. Вона приймає об’єкт типу PreparedStatement та має встановити всі параметри запиту. Основною мотивацією для параметризації запитів в SingleStore є захист від SQL ін’єкцій.
- outputParallelization — логічна змінна, яка вказує чи необхідно перегрупувати результуючу множину для того, щоб розділити її між всіма робітниками. Цей процес необхідний, щоб паралелізувати подальші операції над множиною. За замовчуванням значення параметра істинне.
- rowMapper — об’єкт, який реалізовує інтерфейс RowMapper, а саме функцію mapRow, яка приймає ResultSet та перетворює рядок результату до бажаного типу.

Параметр dataSourceConfiguration є обов’язковим. Параметри query та table є взаємновиключними, але хоча б один з них повинен бути вказаний.

Приклад непаралельного читання даних:

```
PCollection<USER_DATA_TYPE>items = pipeline.apply(
  SingleStoreIO.<USER_DATA_TYPE>read()
    .withDataSourceConfiguration(dc)
    .withTable("MY_TABLE") // or.withQuery("QUERY")
    .withStatementPreparator(statementPreparator)
    .withOutputParallelization(true)
    .withRowMapper(mapper)
);
```

Під час створення технічної документації досліджено 3 способи реалізації паралельного читання даних. Один з них вже був описаний в 1 розділі (підрозділ — формулювання вимог до конектора). Цей спосіб полягає в тому, щоб замість одного SELECT запиту запустити декілька паралельно. Кожен з них

відповідає за прочитання рядків з певного проміжку. Проміжки визначаються за допомогою однієї з колонок результату.
Наприклад замість запиту
SELECT a, b FROM t
конектор може надіслати наступні запити до бази даних
SELECT a, b FROM t WHERE $0 < a < 100$
SELECT a, b FROM t WHERE $100 < a < 200$
SELECT a, b FROM t WHERE $200 < a < 300$
Цей спосіб реалізований в JdbcIO та має наступні недоліки:

- користувач змушений вказати колонку для розбиття запиту
- користувач змушений пересвідчитись, що колонка має індекс, інакше запит буде надто повільним
- один під-запит буде читати інформацію з різних листків SingleStore і через це збільшується кількість трафіку між вузлами розподіленої бази даних
- необхідно або вказати нижню та верхню межу, або конектор буде витрачати ресурси на їх визначення
- для деяких запитів база даних буде виконувати однакові підрахунки для кожного підзапиту.

Головною перевагою цього методу є те, що можна керувати рівнем паралелізму.

SingleStore має спеціальний функціонал для паралельного читання даних. Рекомендований спосіб реалізації паралельного читання складається з наступних кроків:

- Виконати запит CREATE RESULT TABLE tmp AS SELECT ... FROM ...
- Одночасно почати читати дані з кожного розділу бази даних за допомогою запиту SELECT ... FROM ::tmp WHERE partition_id() = id
- Після закінчення цього процесу виконати запит DROP RESULT TABLE tmp.

Крім цього, з'єднання за допомогою якого виконано перший крок весь час має залишатись відкритим. Повторне читання із одного розділу є неможливим [19].

На жаль, цей спосіб суперечить вимогам, які накладає модель Apache Beam. Відповідно до рекомендацій по розробці конекторів, PTransform повинен відповідати наступним вимогам:

- Він повинен підтримувати можливість серіалізації. Це необхідно, щоб клас можна було надіслати системі, яка власне виконує всі операції.
- Він повинен бути незмінним. Всі приватні поля класу повинні бути оголошені як final.
- Він має підтримувати можливість паралельного виконання. При цьому обчислення над різних розділах даних повинні бути незалежними. Також, при виникненні помилок процес має мати можливість повторного виконання.
- Його має бути можливо покрити юніт-тестами [8].

Перша проблема виникає із серіалізацією. Об'єкт з'єднання з базою даних не можна серіалізувати. Для створення таблиці з результатом потрібно підтримувати відкрите з'єднання впродовж всього часу читання. Такі з'єднання можна було б зберігати незалежно від PTransform. Але тоді всі паралельні процеси будуть залежати від одного об'єкта, що порушує третю умову.

Наступна проблема це те, що всі паралельні процеси читання мають почати читати одночасно. Beam не надає таких гарантій. Звісно, якщо кластер на якому виконуються операції має достатньо ресурсів, то він запустить всі процеси. Але конектор не має можливостей, щоб перевірити чи ця умова виконується. Через це відповідальність за забезпечення конектора достатньою кількістю ресурсів для виконання читання лягає на користувача. Це значно погіршує досвід використання конектора. До прикладу, якщо користувач захоче одночасно запустити читання двох таблиць за допомогою Spark, то він має пересвідчитись що кількість ядер в Spark кластері більша за кількість розділів в базі даних помноженій на два. Часто користувачі не мають достатньо глибокого розуміння принципів роботи зазначених технологій, щоб забезпечувати необхідні гарантії. При цьому, це руйнує основну перевагу моделі Beam — простоту використання та незалежність від системи, що виконує обчислення.

Третьою проблемою є те, що читання не можна виконати повторно при виникненні помилок. Якщо виникла помилка при читанні одного розділу — весь процес потрібно повторити з самого початку. При збільшенні кількості даних до певної межі виникає ситуація, що читання хоча б з одного розділу завжди зазнає помилки. Це може ставатись через помилки із з'єднанням або нестабільність мережі. Якщо щоразу перезапускати весь процес читання, то він може ніколи не закінчитись.

SingleStore підтримує читання з матеріалізованих таблиць результату. Відмінністю цих таблиць від попередніх є те, що з матеріалізованої таблиці можна читати один розділ декілька разів, а також що не обов'язково починати читати всі розділи одночасно.

Хоча такий спосіб і вирішує деякі проблеми, але всерівно виникає необхідність зберігати відкрите з'єднання впродовж часу читання. Ще матеріалізовані таблиці результату зберігають всі данні в оперативній пам'яті бази даних. Для читання великих результатів (терабайти данних) цей підхід не підійде.

Одним із завдань є інтегрувати конектор в GitHub репозиторій Apache Beam, щоб в майбутньому за його підтримку відповідали розробники Apache Beam. Розглянутий спосіб паралельного читання порушує рекомендації до розробки конекторів, а отже інтегрувати його буде надзвичайно важко.

Розглянемо ще один спосіб паралельного читання даних. Припустимо, що користвач хоче прочитати результат SELECT запиту. Цей запит можна обгорнути в ще один SELECT запит та додати фільтр WHERE partition_id()=x. Таким чином, можна прочитати рядки тільки з одного розділу бази даних. Такі запити можна легко запустити для кожного розділу. Вони будуть повністю незалежними.

Наприклад, замість запиту:
SELECT a, b FROM t
конектор може надіслати наступні запити до бази даних:
SELECT a, b FROM t WHERE partition_id() = 0
SELECT a, b FROM t WHERE partition_id() = 1
SELECT a, b FROM t WHERE partition_id() = 2
SELECT a, b FROM t WHERE partition_id() = 4

Перевагами цього методу порівняно із першим запропонованим є те, що:

- користувач не змушений вказати колонку для розбиття запиту і не має

хвилюватись що ця колонка має індекс
- один під-запит буде читати інформацію з одного листка SingleStore і через це кількість трафіку між вузлами розподіленої бази даних є мінімальною
- користувач не повинен вказувати кількість розділів результату та верхню/нижню межі колонок

Отже, реалізувавши паралельне читання за допомогою цього способу значно спроститься конфігурація та використання конектора. Саме цей варіант вибраний для фінальної реалізації.

Паралельне читання можна конфігурувати за допомогою наступних параметрів:
- dataSourceConfiguration — об'єкт типу DataSourceConfiguration з усією інформацією необхідною для створення з'єднання
- query — SQL запит, що має бути виконаний
- table — таблиця з якої читати дані
- rowMapper - об'єкт, який реалізовує інтерфейс RowMapper, а саме функцію mapRow, яка приймає ResultSet та перетворює рядок результату до бажаного типу.

Приклад паралельного читання даних:
```
PCollection<USER_DATA_TYPE>items = pipeline.apply(
   SingleStoreIO.<USER_DATA_TYPE>readWithPartitions()
      .withDataSourceConfiguration(dc)
      .withTable("MY_TABLE") // or.withQuery("QUERY")
      .withRowMapper(mapper)
);
```

Для запису даних елементи множини необхідно розбити на групи. Кожна група надсилається до бази даних за допомогою окремого LOAD DATA запиту. На відміну від INSERT запитів, які використовуються в JdbcIOконекторі, вони надсилають дані як окремий потік байтів, а не як частину запиту. За рахунок цього, дані можна відправляти набагато більшими групами. Також, зменшується кількість трафіку.

PTransform для запису даних підтримує наступні параметри:
- dataSourceConfiguration — об'єкт типу DataSourceConfiguration з усією інформацією необхідною для створення з'єднання
- table — таблиця в яку записувати дані
- batchSize — Кількість елементів множини, які надстлаються одним запитом. За замовчуванням — 100000.
- userDataMapper — об'єкт, який реалізовує інтерфейс UserDataMapper, а саме функцію mapRow, яка приймає елемент множини та перетворює масив стрічок. Ці стрічки потім об'єднуються в CSV формат та надсилаються до бази даних

Приклад запису даних:
```
data.apply(
   SingleStoreIO.<USER_DATA_TYPE>write()
      .withDataSourceConfiguration(dc)
      .withTable("MY_TABLE")
      .withUserDataMapper(mapper)
      .withBatchSize(100000)
```

);

Проведено тестування швидкодії запису даних за допомогою реалізованого конектора та існуючого JdbcIO. Під час тестування базу даних SingleStore, запущену в docker контейнері. Результати тестування наведені в таблиці 1. Відповідно до них, LOAD DATA запити працюють приблизно вдвічі швидше ніж INSERT запити.

Таблиця 1.
Порівняння швидкодії запису даних використовуючи SingleStoreIO та JdbcIO.

Кількість рядків	**SingleStoreIO**	**JdbcIO**
100000	4.638 сек.	6.772 сек.
200000	8.312 сек.	12.269 сек.
500000	17.956 сек.	38.105 сек.
1000000	33.877 сек.	78.197 сек.

Для того, щоб конектор використовували, його потрібно протестувати, задокументувати та поширити на платформах, де користувачі зможуть легко про нього дізнатись. У випадку Beam, найкращим способом поширити конектор є інтегрувати його в GitHub репозиторій Apache Beam. Конектори, які інтегровані в цей GitHubрепозиторій називаються вбудованими (built-in). Їхньою підтримкою займаються ком'юніті Beam, що є великою перевагою, адже не потрібно буде витрачати ресурси на виправлення помилок, підтримування інтегрованої системи тестування. Також, ці конектори можуть мати свої web-сторінки в документації Apache Beam.

Для інтеграції конектора в репозиторій Apache Beam, він повинен бути протестований відповідно до їхніх стандартів [7].

Тести запускаються на платформи Jenkins(https://www.jenkins.io/) за допомогою GitHubActions(https://github.com/features/actions).

Для конектора написано наступні види тестів:

- Юніт тести — тести, що виконують невеликі частини коду конектора. Вони використовують фреймворкJUint (https://junit.org/junit5/). Ці тести комунікують з макетом бази даних створеним за допомогою фреймворкуmockito. Ця група тестів запускається кожного разу, коли хтось створює гілку і змінює в ній код пов'язаний із певним тестом.
- Інтеграційні тести. Ці тести комунікують із справжньою базою даних і перевіряють коректність основного функціоналу. SingleStore запускається за допомогою технології Kubernetes (https://docs.singlestore.com/db/v8.0/en/deploy/kubernetes.html). Цю групу тестів запускають розробники Apache Beam перед тим, як додати зміни в головну гілку.
- Тести продуктивності. Ці тести як і інтеграційні тести комунікують із справжньою базою даних, але вони завантажують великі масиви даних для вимірювання продуктивності конектора. Ця група тестів запускається раз в день для того, щоб перевірити чи якісь зміни не спровокували погіршення продуктивності.

Тести продуктивності запускаються раз в день і їхній результат візуалізується за допомогою Grafana.

Обговорення результатів дослідження. Перед розробкою конектора створено GitHub завдання з пропозицією розробки SingleStoreIO конектора (https://github.com/apache/beam/issues/22617) проведено консультацію з розробниками компанії Google (LukaszCwik, KennethKnowles, BrianHulette, ChamikaraJayalath, JohnCasey) та розроблено технічну документацію. В цій документації узгоджені всі проблемні питання реалізації конектора. На основі даного документу здійснені зміни в коді, які представлені у вигляді двох пул реквестів до репозиторію Apache Beam:

- https://github.com/apache/beam/pull/23535
- https://github.com/apache/beam/pull/24290

Всі зміни протестовано. Перевіркою коду займались наступні розробники:

- JohnCasey(https://github.com/johnjcasey)
- YiHu(https://github.com/Abacn)
- AhmadAbualsaud(https://github.com/ahmedabu98)
- PabloEstrada(https://github.com/pabloem)

Їхні коментарі були враховані і після успішного проходження тестів код додано в головну гілку проєкту. В результаті, підтримка бази даних SingleStore додана в Apache Beam у версії 2.44 (https://beam.apache.org/blog/beam-2.44.0/)

Конектор можна завантажити або додати в систему автоматичного збирання із Maven репозиторію (https://mvnrepository.com/artifact/org.apache.beam/beam-sdks-java-io-singlestore).

Документування бібліотеки є надзвичайно важливим етапом. Без нього користувачі можуть просто не дізнатись про її існування або не могти коректно використовувати. Apache Beam розробили стандарти для документування конекторів [10]. На момент створення конектора ці стандарти ще були на стадії розробки і нами використовувався доступ до робочого варіанту.

Весь текст документації зберігається в тому самому GitHub репозиторії, що й код конекторів. Для оновлення документації створено наступний пул реквест (https://github.com/apache/beam/pull/24377). Його перевірено та додано до головної гілки YiHu (https://github.com/Abacn/).

Дані зміни включали створення сторінки, яка містить пояснення всіх перетворень доданих конектором, їх параметрів та прикладів використання [2]. Також SingleStoreIO конектор додано до таблиці всіх вбудованих конекторів [16]. Автори документації SingleStore теж оновили її, додавши інформацію про цей конектор [21]. Через деякий час після релізу конектора Akmal Chaudhri (старший технічний євангеліст компанії SingleStore) написав блог про використання цього конектора [2].

Наукова новизна та практична цінність дослідження. Конектор може бути використаний для ETL процесів під час яких Apache Beam відповідає за обробку та збереження даних в SingleStoreDB. Він надає зручніший спосіб конфігурації та кращу продуктивність в порівнянні із JdbcIO конектором. Також, результати досліджень можуть використовуватись для створення конекторів між іншими технологіями та СКБД SingleStoreDB.

3. Висновки та перспективи подальших досліджень. На основі аналізу можливих сценаріїв застосування Apache Beam та SingleStore та особливостей цих технологій сформульовано вимоги до конектора необхідні для задоволення потреб користувачів, а саме:

- можливість паралельного запису даних за допомогою LOAD DATA;
- можливість паралельного читання даних за допомогою розподілених властивостей SingleStoreDB;
- можливість виконати запити читання даних непаралельно для специфічних запитів.

Розроблено конектор, який відповідає наведеним вимогам. Під час розробки досліджено можливість використання різних способів паралельного читання та запису даних. Згідно досліджень найоптимальнішим способом паралельного читання є розбиття запиту на декілька незалежних використовуючи певну колонку. Найоптимальнішим способом запису даних є використання запиту LOAD DATA. Конектор протестовано та налаштована система неперервної інтеграції.

Код конектора перевірено спеціалістами та інтегровано в GitHub репозиторій Apache Beam. Вихідний код конектора можна знайти за посиланням : (https://github.com/apache/beam/tree/master/sdks/java/io/singlestore). Він надає зручніший спосіб конфігурації та кращу продуктивність в порівнянні із JdbcIO конектором. Інформацію про SingleStore конектор можна знайти на офіційній документації Apache Beam [26] та офіційній документації SingleStore [16]. Перспективами подальших досліджень є спроби реалізації конекторів для аналогічних технологій таких як Hadoop, Flume, Flink.

Список використаної літератури

1. Adalbert M. SingleStore Apache Beam IO Connector Design Doc. Retrieved from: https://docs.google.com/document/d/1WUhkoZ93SaGXyOz_UtX0jXzIRl194hCId_IdmEV9jw/edit#heading=h.wskna8eurvjv
2. Akmal Ch. Quicktip: Using Apache Beam with SingleStoreDB. Retrieved from: https://medium.com/@VeryFatBoy/quick-tip-using-apache-beam-with-singlestoredb-452d4256a7ca
3. Apache Beam Overview. Retrieved from: https://beam.apache.org/get-started/beam-overview/
4. Automate your workflow from idea to production. Retrieved from: https://github.com/features/actions
5. Basics of the Beam model. Retrieved from: https://beam.apache.org/documentation/basics/
6. Cluster Components. Retrieved from: https://docs.singlestore.com/db/v8.0/en/introduction/distributed-architecture/cluster-components.html#cluster-components
7. Contribution Testing Guide. Retrieved from: https://cwiki.apache.org/confluence/display/BEAM/Contribution+Testing+Guide
8. Developing I/O connectors for Java. Retrieved from: https://beam.apache.org/documentation/io/developing-io-java/
9. [Template] Apache Beam Design Doc. Retrieved from: https://docs.google.com/document/d/1kVePqjt2daZd0bQHGUwghlcLbhvrny7VpflAzk9sjUg/edit?usp=sharing
10. I/O Standards. Retrieved from: https://beam.apache.org/documentation/io/io-standards/
11. Interaction of Cluster Components. Retrieved from: https://docs.singlestore.com/db/v8.0/en/introduction/distributed-architecture/interaction-of-cluster-components.html#interaction-of-cluster-components
12. Jeffrey Deanand Sanjay Ghemawat. MapReduce: Simplified Data Processingon Large Clusters. To appear in OSDI 2004. Retrieved from: https://static.googleusercontent.com/media/research.google.com/uk//archive/mapreduce-osdi04.pdf
13. Jenkins. Retrieved from: https://www.jenkins.io/
14. JosephVictor. Engineering. The future is bottomless. Retrieved from: https://www.singlestore.com/blog/the-future-is-bottomless/

15. Overview: Developing a new I/O connector. Retrieved from: https://beam.apache.org/documentation/io/developing-io-overview/
16. I/O Connectors. Retrieved from: https://beam.apache.org/documentation/io/connectors/
17. SingleStore Documentation. Retrieved from: https://docs.singlestore.com/managed-service/en/reference/sql-reference/data-manipulation-language-dml/insert.html
18. SingleStore LOAD DATA. Retrieved from: https://docs.singlestore.com/managed-service/en/reference/sql-reference/data-manipulation-language-dml/load-data.html
19. SingleStore Read Query Results in Parallel. Retrieved from: https://docs.singlestore.com/managed-service/en/query-data/read-query-results-in-parallel.html
20. SingleStore. Retrieved from: https://download.oracle.com/otn-pub/jcp/jdbc-4_2-mrel2-spec/jdbc4.2-fr-spec.pdf?AuthParam=1683632646_268549c0a0b191629570c83d4e48b211
21. SingleStore. Load Data from Apache Beam. Retrieved from: https://docs.singlestore.com/managed-service/en/load-data/load-data-from-a-data-source/load-data-from-apache-beam.html
22. THE APACHE SOFTWARE FOUNDATION ANNOUNCES APACHE® BEAM™ AS A TOP-LEVEL PROJECT. Retrieved from: https://news.apache.org/foundation/entry/the-apache-software-foundation-announces (дата звернення 17.04.2023)
23. The SingleStore JDBC Driver. Retrieved from:https://docs.singlestore.com/managed-service/en/developer-resources/connect-with-application-development-tools/connect-with-java-jdbc/the-singlestore-jdbc-driver.html#connection-string-parameters
24. Universal Storage. Retrieved from: https://docs.singlestore.com/db/v8.0/en/create-a-database/columnstore/universal-storage.html
25. Why Apache Beam is the next big thing in big data processing. Retrieved from: https://medium.com/@shafiqa_iqbal/why-apache-beam-is-the-next-big-thing-in-big-data-processing-808abacf52f1
26. SingleStoreDB I/O. Retrieved from: https://beam.apache.org/documentation/io/built-in/singlestore/

Hlebena M. I., Makarovych A. V. SingleStoreDB connector for Apache Beam.

The process of developing the SingleStoreDB connector for Apache Beam is described, along with the challenges encountered during this process. Java programming language was utilized for implementing the connector. Gradle, an automated build system, was used for project compilation and dependency management. Continuous testing of the connector was ensured using the Jenkins continuous integration tool. Communication with the database was facilitated using the SingleStore JDBC Driver library. The research object selected was the data exchange process between Apache Beam and SingleStoreDB, with the connector serving as the subject of investigation, enabling efficient data exchange between the two platforms. Requirements analysis was conducted for the connector, identifying key requirements such as the ability for parallel data writes, parallel data reads using SingleStoreDB distributed properties, and the ability to execute non-parallel data read queries for specific requests. Various methods of parallel data reading and writing were explored, with research indicating that the most optimal method for parallel reading involves splitting the query into several independent ones using a certain column. The most efficient method for data writing was found to be using the LOAD DATA query, which sends data as a separate stream of bytes rather than as part of the query, allowing for larger data batches to be sent. The connector was tested, and continuous integration was set up. Documentation was developed, and the connector was integrated into the Apache Beam GitHub repository. The developed connector can be used for ETL processes where Apache Beam is responsible for data processing and storage in SingleStoreDB, providing a more convenient configuration and better performance compared to the JdbcIO connector. The research results can be utilized for creating connectors between other technologies and the SingleStoreDB database management system.

Keywords: database, connector, distributed systems, ETL, Apache Beam, SingleStore, OLAP, OLTP.

References

1. Adalbert, M. SingleStore Apache Beam IO Connector Design Doc. Retrieved from: https://docs.google.com/document/d/1WU-hkoZ93SaGXyOz_UtX0jXzIRl194hCId_IdmEV9jw/edit#heading=h.wskna8eurvjv
2. Akmal, Ch. Quicktip: Using Apache Beam with SingleStoreDB. Retrieved from: https://medium.com/@VeryFatBoy/quick-tip-using-apache-beam-with-singlestoredb-452d4256a7ca
3. Apache Beam Overview. Retrieved from: https://beam.apache.org/get-started/beam-overview/
4. Automate your workflow from idea to production. Retrieved from: https://github.com/features/actions
5. Basics of the Beam model. Retrieved from: https://beam.apache.org/documentation/basics/
6. Cluster Components. Retrieved from: https://docs.singlestore.com/db/v8.0/en/introduction/distributed-architecture/cluster-components.html#cluster-components
7. Contribution Testing Guide. Retrieved from: https://cwiki.apache.org/confluence/display/BEAM/Contribution+Testing+Guide
8. Developing I/O connectors for Java. Retrieved from: https://beam.apache.org/documentation/io/developing-io-java/
9. [Template] Apache Beam Design Doc. Retrieved from: https://docs.google.com/document/d/1kVePqjt2daZd0bQHGUwghlcLbhvrny7VpflAzk9sjUg/edit?usp=sharing
10. I/O Standards. Retrieved from: https://beam.apache.org/documentation/io/io-standards/
11. Interaction of Cluster Components. Retrieved from: https://docs.singlestore.com/db/v8.0/en/introduction/distributed-architecture/interaction-of-cluster-components.html#interaction-of-cluster-components
12. Jeffrey Deanand Sanjay Ghemawat. MapReduce: Simplified Data Processingon Large Clusters. To appear in OSDI 2004. Retrieved from: https://static.googleusercontent.com/media/research.google.com/uk//archive/mapreduce-osdi04.pdf
13. Jenkins. Retrieved from: https://www.jenkins.io/
14. JosephVictor. Engineering. The future is bottomless. Retrieved from: https://www.singlestore.com/blog/the-future-is-bottomless/
15. Overview: Developing a new I/O connector. Retrieved from: https://beam.apache.org/documentation/io/developing-io-overview/
16. I/O Connectors. Retrieved from: https://beam.apache.org/documentation/io/connectors/
17. SingleStore Documentation. Retrieved from: https://docs.singlestore.com/managed-service/en/reference/sql-reference/data-manipulation-language-dml/insert.html
18. SingleStore LOAD DATA. Retrieved from: https://docs.singlestore.com/managed-service/en/reference/sql-reference/data-manipulation-language-dml/load-data.html
19. SingleStore Read Query Results in Parallel. Retrieved from: https://docs.singlestore.com/managed-service/en/query-data/read-query-results-in-parallel.html
20. SingleStore. Retrieved from: https://download.oracle.com/otn-pub/jcp/jdbc-4_2-mrel2-spec/jdbc4.2-fr-spec.pdf?AuthParam=1683632646_268549c0a0b191629570c83d4e48b211
21. SingleStore. Load Data from Apache Beam. Retrieved from: https://docs.singlestore.com/managed-service/en/load-data/load-data-from-a-data-source/load-data-from-apache-beam.html
22. THE APACHE SOFTWARE FOUNDATION ANNOUNCES APACHE® BEAM™ AS A TOP-LEVEL PROJECT. Retrieved from: https://news.apache.org/foundation/entry/the-apache-software-foundation-announces
23. The SingleStore JDBC Driver. Retrieved from: https://docs.singlestore.com/managed-service/en/developer-resources/connect-with-application-development-tools/connect-with-java-jdbc/the-singlestore-jdbc-driver.html#connection-string-parameters
24. Universal Storage. Retrieved from: https://docs.singlestore.com/db/v8.0/en/create-a-database/columnstore/universal-storage.html
25. Why Apache Beam is the next big thing in big data processing. Retrieved

from: https://medium.com/@shafiqa_iqbal/why-apache-beam-is-the-next-big-thing-in-big-data-processing-808abacf52f1
26. SingleStoreDB I/O. Retrieved from: https://beam.apache.org/documentation/io/built-in/singlestore/

Одержано 20.04.2024