ISSN 2224-087X. Електроніка та інформаційні технології. 2020. Випуск 13. С. 69–76
Electronics and information technologies. 2020. Issue 13. P. 69–76

УДК 338.22 DOI: https://doi.org/10.30970/eli.13.7

МЕТОД ГОРИЗОНТАЛЬНОГО МАСШТАБУВАННЯ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ

О. Семочко, І. Половинко

Львівський національний університет імені Івана Франка,
вул. Ген. Тарнавського, 107, 79017 Львів, Україна
semochko@gmail.com , polovynkoi@gmail.com

В роботі розглянуто метод використання бізнес-аналізу у виборі оптимального рішення для практичного застосування платформи розподілення потоків. У даному методі використано відкрите програмне забезпечення Apache Kafka, яке розроблене компанією LinkedIn та передане Apache, для вільного (безкоштовного) ліцензування. Розглянуто масштабування паралельно великої кількості одночасних запитів. Обговорюється також використання даної платформи для обробки поточних завдань для систем що нараховують більше 2 млн. користувачів.

Ключові слова: бізнес аналіз, архітектурні рішення, Apache Kafka, Apache Connector, Apache Streams, горизонтальне масштабування, розподілені обчислення, паралельні обчислення.

Вступ

Потреба у бізнес-аналізі виникла у компаній, які зіткнулись з проблемою оновленням та модифікацією існуючих інформаційних систем. Основні задачі, які ставились перед бізнес аналітиками – це зменшити витрати та підвищити ефективність роботи, здатність до масштабування та впровадження сучасних цифрових технологій [1].

До цього часу існуючі системи масштабувалась в основному лише завдяки збільшенню кількості обладнання. Однак, при такому підході, перестає працювати закон Мура [2], який передбачає експоненціальне зростання ефективності. Крім того, вказаний підхід не дає можливості на належному рівні здійснювати паралельні обчислення. Тому, метою даної роботи було змінити архітектуру інформаційної системи з максимальним збереження існуючої структури. На рис. 1 показано схему діяльності бізнес-аналізу.

Вхідні дані - це документація клієнта, до якої відносяться вимоги бізнесу до продукту чи сервісу. Їх можна отримати у вигляді як опису, так і під час подальших інтерв'ю з експертами замовника. *Пропозиція Архітекторів* – рішення або комплекс програмних рішень, які можуть бути використані для задоволення потреб продуктів чи сервісів бізнесу. *Бюджет та Обмеження* – фактори, що прямо впливають на вибір кращого рішення. До *Обмежень* відноситься існуюча архітектура (бази даних, мова програмування існуючого рішення, програмне забезпечення серверів, конфігурація серверного обладнання).

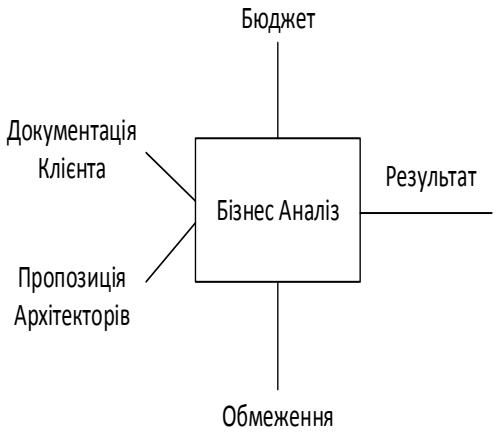

Рис.1. Вхідні та вихідні даних задач бізнес-аналізу.

Із рис.1 видно, що кінцевим продуктом розв'язку задачі бізнес аналізу є *Результат* – мета бізнес аналізу обрати найкраще рішення для бізнесу, на основі вхідних даних, враховуючи фактори, що прямо або опосередковано впливають.

У роботі для обробки великої кількості даних (Big data) використано високонавантажену систему (Highload System) [3]. Така система відповідає наступним критеріям:

- велика кількість користувачів, що здійснюють одночасні певні дії;
- великий об'єм даних, який необхідно опрацьовувати;
- багаточисельні та складні операції;
- висока швидкість відповідної системи;
- здатність до масштабування;
- модульність.

Для оцінки величини навантаження системи використовують показники пропускної здатності (throughput) та затримки [3]. Вони визначають з якою максимальною швидкістю дані можуть бути опрацьовані системою та час, який система затрачає на збереження чи отримання даних відповідно. Для того, щоб зберегти баланс між затратами на вартість з одного боку, та вимогами щодо пропускної здатності і затримкою з іншого боку, задаються наступні вимоги до системи:

- висока пропускна здатність до 1 Гбіт/с, і при цьому низька затримка (до 100 мс) (для достовірності показників, заміри можна проводити на найбільш затратному та складному процесі – моніторингу доступності сервісів у системі);
- висока пропускна здатність (максимально можлива) та достатньо висока затримка, але не більше 200 мс, при обробці об'єктів у реальному часі, наприклад, при шифруванні даних користувачів для забезпечення їх конфіденційності;
- низька пропускна здатність та низька затримка, до 10 мс, наприклад, перевірка введення даних (орфографія, гнучкий пошук тощо) у полях форм користувачів.

Для масштабування системи скористаємось підходом, що базується на законі Амдала [4], який ілюструє обмеження росту продуктивності обчислювальної системи із збільшенням кількості обчислювачів. Згідно з цим законом, у випадку, коли задача розподіляється на декілька частин, сумарний час її виконання на паралельній системі не може бути меншим від часу виконання найбільш повільного фрагменту. При цьому прискорення виконання програми за рахунок розпаралелення її інструкцій на множину обчислювачів обмежене часом, необхідним на виконання її послідовних інструкцій. Якщо алгоритм задачі допускає, що частина α від загального об'єму може бути отримана лише послідовними розрахунками, тоді частина $1-\alpha$ програми може бути розпаралелена ідеально. Тобто час обчислення буде обернено пропорційний числу задіяних вузлів *p*. Тоді прискорення, яке може бути отримане на обчислювальній системі, що складається з *p* процесорів, у порівнянні з однопроцесорним рішенням, не буде перевищувати величини:

$$S == \frac{1}{\alpha + \frac{1-\alpha}{p}}.$$

У таблиці показано збільшення швидкодії програми (у рази) із часткою послідовних обчислень α при використанні *p* процесорів.

α / *p*	10	100	1000
0	10	100	1000
0,1	5,263	9,174	9,910
0,25	3,077	3,883	3,988
0,4	2,174	2,463	2,496

З таблиці видно, що лише алгоритм, який не містить послідовних обчислень ($\alpha = 0$) дозволяє отримати лінійний приріст продуктивності з ростом кількості обчислювачів у системі. Якщо ж доля послідовних обчислень в алгоритмі рівна 25%, то збільшення числа процесорів до 10 дає прискорення лише у 3,077 разів. Отже при частці послідовних обчислень α загальний приріст продуктивності не може перевищувати $1/\alpha$. Отже, якщо половина коду послідовна, то загальний приріст ніколи не перевищить 2. Закон Адамара також показує, що приріст ефективності обчислень залежить від алгоритму задачі і завжди обмежений зверху, якщо $\alpha \neq 0$. Отже не для всякої задачі є сенс нарощувати число процесорів у обчислювальній системі. Крім того, якщо врахувати час, необхідний для передачі даних між вузлами обчислювальної системи, то може виявитись, що залежність часу обчислень від числа вузлів буде мати мінімум. Це накладає обмеження на масштабованість обчислювальної системи. Тобто, починаючи з певного моменту, додавання нових вузлів у систему буде збільшувати час, необхідний для розв'язку задачі.

Результат роботи

У роботі використано метод горизонтального масштабування який полягає у розділенні системи на окремі структурні компоненти, часто з рознесенням їх по різних

серверах, які можуть паралельно виконувати окремі функції. Цей метод потребує внесення зміни програмного коду у роботі програми, при цьому залишається обов'язковим здійснення роботи програми як однієї цілісної системи.

Реалізація завдання здійснювалась за допомогою системи Apache Kafka, яка дозволяє не виходити за рамки встановлених показників, та використати принцип горизонтального масштабування [5]. Така система реалізована у реально працюючих бізнес-проектах. Зокрема, американська щоденна газета The New York Times використовує Apache Kafka Streams для зберігання і поширення в реальному часі опублікованого контенту серед різних додатків і систем, які роблять його доступним для читачів (сервіси підписки, мобільні додатки, новинні агрегатори та ін.);соціальна мережа фотозображень Pinterest за допомогою Apache Kafka. Кафка Стрімс змогла масштабувати власну Інтелектуальну систему бюджетування своєї рекламної інфраструктури в режимі реального часу, підвищивши точність прогнозів по фінансових витратах. Один з найбільших банків Нідерландів, Rabobank застосовує Кафка Стрімс для оповіщення клієнтів в режимі реального часу про фінансові події.

Принцип її роботи полягає у підвищенні пропускної здатності завдяки збільшенню розміру пакету (batch) запитів. Під пакетом будемо розуміти групу запитів, від продюсерів (producers), що відправляються як одне ціле, для зберігання в одному розділі (partition). Продюсери відповідають за те, який саме пакет використати для запису до конкретного розділу. Вони працюють за принципом циклічного планування (round-robin). У загальному випадку цей алгоритм описує планування процесів та комутації пакетів даних. В даному випадку він використовується як диспетчер пакетів даних для балансу навантаження або визначає як це можна зробити відповідно до певної функції семантичного розділення (скажімо, на основі якогось ключа в записі).

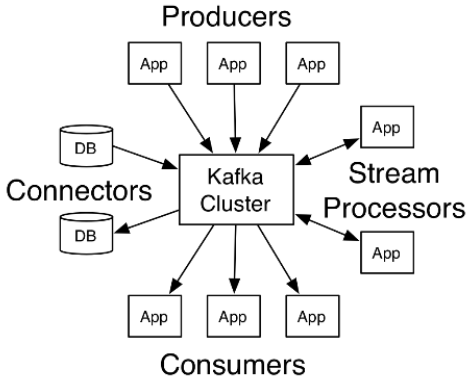

Рис. 2. Типова архітектура роботи Apache Kafka [5].

На рис.2 представлено типову архітектуру роботи Apache Kafka. Тут *Kafka Cluster* – це платформа з розподіленими потоками, що об'єднуються в теми (topics); *Consumers* (консюмери) – дозволяє додатку (групі додатків) підписатися на одну або декілька тем (topics) і обробляти потік записів, створених до них; *Connectors* (конектори) – дозволяють створювати та запускати багаторазові продюсери або консюмери, які підключають Kafka до існуючих додатків або систем даних. Наприклад, конектор до

реляційної бази даних може фіксувати всі зміни в таблиці; *Producers* (продюсери) – дозволяє додатку (групі додатків) публікувати потік записів із однієї або декількох тем (topics) Kafka; *DB* – реляційні бази даних; *App* – додаток або група додатків;

З рис. 2 випливає, що у випадку коли продюсери виробляють більше запитів одного типу, то число консюмерів (consumers) буде відповідно збільшуватись. Консюмери використовуються для зчитування даних з розділу, і можуть об'єднуватись в групи консюмерів.

На рис. 3, наведено приклад об'єднання консюмерів в групи, на основі роботи з двома серверами, котрі мають 4 різних розділи для роботи з двома групами консюмерів, різного розміру. При тому, в групі консюмерів А працює 2 представники (instance), а в групі консюмерів В працює 4 представника (instance). При тому, обробка їх потоків даних здійснюється перехресно з різними серверами та різними розділами.

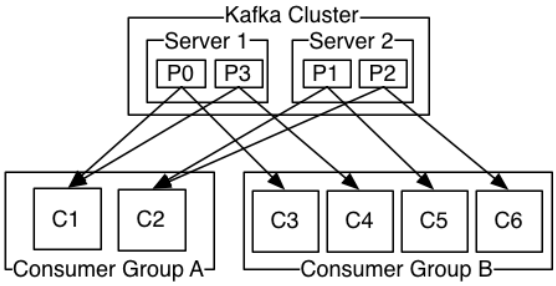

Рис. 3. Схема роботи об'єднаних груп консюмерів.

Усі функції, що описані вище, виконує розроблена авторами оригінальна програма обробки текстових масивів (Array) та стрічок (String) підключення Streams API у Java 8+ для роботи з BigData. Програма представлена в додатку.

Висновки

Проведений у роботі аналіз показав, що зміна параметру розміру пакету (batch.size) може підвищити пропускну здатність, зменшити навантаження при обробці пакетів а також операцій потокового введення-виведення даних. А при низькому навантаженні Кафка може збільшувати затримку відправки пакетів, оскільки продюсер буде очікувати готовності пакету.

На основі отриманих результатів також можна стверджувати, що не достатньо лише забезпечити зчитування та запис потоку даних, головна мета – це забезпечити обробку потоків у реальному часі. Для цього можна використати Kafka Streams API – потоковий процесор, що приймає безперервні потоки даних з вхідних каналів, виконує певну обробку на цьому вході, та забезпечує безперервне виведення даних.

Отже, завдяки новій архітектурі, вдалося реалізувати можливість масштабування без збільшення кількості обладнання. При цьому здійснено переведення системи на новий принцип роботи – модульність (оскільки Кафка дозволяє підключати модулі та аплікації незалежно від мови програмування), та забезпечити подальшу ескалацію бізнесу клієнта.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. A guide to the business analysis body of knowledge (BABOK) v3 - International Institute of Business Analysis (IIBA), ISBN-13: 978-1-927584-03-3 – Toronto, Ontario, Canada – 2015
2. *Gordon E.* Moore Cramming more components onto integrated circuits" - Electronics, Volume 38, Number 8, April 19, 1965 [Електронний ресурс] – Режим доступу https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf
3. HighLoad++ для начинающих [Електронний ресурс] – Режим доступу http://highload.guide/blog/highload-forbeginners.html
4. *Amdahl G.* «The validity of the single processor approach to achieving large-scale computing capabilities». In Proceedings of AFIPS Spring Joint Computer Conference, Atlantic City, N.J., AFIPS Press, pp. 483-85. – April 1967
5. Kafka 2.5 Documentation [Електронний ресурс] – Режим доступу https://kafka.apache.org/documentation/#gettingStarted

Додаток.

Демо програма обробки текстових масивів

```
- import org.apache.kafka.common.serialization.Serdes;
- import org.apache.kafka.common.utils.Bytes;
- import org.apache.kafka.streams.KafkaStreams;
- import org.apache.kafka.streams.StreamsBuilder;
- import org.apache.kafka.streams.StreamsConfig;
- import org.apache.kafka.streams.kstream.KStream;
- import org.apache.kafka.streams.kstream.KTable;
- import org.apache.kafka.streams.kstream.Materialized;
- import org.apache.kafka.streams.kstream.Produced;
- import org.apache.kafka.streams.state.KeyValueStore;
-
- import java.util.Arrays;
- import java.util.Properties;
-
- public class WordCountApplication {
-
-     public static void main(final String[] args) throws
  Exception {
-         Properties props = new Properties();
-         props.put(StreamsConfig.APPLICATION_ID_CONFIG,
  "word-count-application");
-         props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
  "kaf-ka-broker1:9092");
```

```
-
   props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
   Serdes.String().getClass());
-
   props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
   Serdes.String().getClass());
-
-          StreamsBuilder builder = new StreamsBuilder();
-          KStream<String, String> textLines = build-
   er.stream("TextLinesTopic");
-          KTable<String, Long> wordCounts = textLines
-              .flatMapValues(textLine -> Ar-
   rays.asList(textLine.toLowerCase().split("\\W+")))
-              .groupBy((key, word) -> word)
-              .count(Materialized.<String, Long, Key-
   ValueStore<Bytes, byte[]>>as("counts-store"));
-          wordCounts.toStream().to("WordsWithCountsTopic",
   Produced.with(Serdes.String(), Serdes.Long()));
-
-          KafkaStreams streams = new Kaf-
   kaStreams(builder.build(), props);
-          streams.start();
-      }
-
-  }
```

METHOD OF HORIZONTAL SCALING OF DISTRIBUTED COMPUTINGS IN HIGH-LOAD SYSTEMS

O. Semochko, I. Polovynko

Ivan Franko National University of Lviv,
107 Generala Tarnavskoho, Lviv, Ukraine 79017
semochko@gmail.com , polovynkoi@gmail.com

In this article, discovered the method of use of business analysis in choosing the optimal solution for practical application distribution platform streams. This method uses the open source software Apache Kafka, developed by LinkedIn and provided by Apache, for free (permissive free software) licensing. Considered scaling of a large number of concurrent requests. The use of this platform to handle current tasks for systems with more than 2 million users is also discussed.

The method of horizontal scaling is used in the work, which consists in dividing the system into separate structural components, often with their distribution on different servers, which can perform separate functions in parallel. This method requires a change in the program code in the operation of the program, while it remains mandatory to implement the program as a single integrated system.

The task was implemented using the Apache Kafka system, which allows not to go beyond the established indicators, and to use the principle of horizontal scaling. This system is implemented in real business projects: In particular, the American daily The New York Times uses Apache Kafka Streams to store and distribute real-time published content among various applications and systems that make it available to readers (subscription services, mobile applications, news aggregators, etc.), social network of Pinterest photos using Apache Kafka. Kafka Streams was able to scale its own Intelligent Budgeting System for its advertising infrastructure in real time, increasing the accuracy of financial cost forecasts. One of the largest banks in the Netherlands, Rabobank uses Kafka Streams to notify customers in real time about financial events.

The principle of its operation is to increase throughput by increasing the size of the batch of requests. Under the batch, we mean a group of requests from producers, sent as a whole, for storage in one partition. Producers are responsible for which batch to use to write to a specific partition. They work on the principle of cyclic planning (round-robin). In the general case, this algorithm describes process scheduling and communication of data batch. In this case, it is used as a data batch manager for load balance or determines how this can be done according to a certain semantic separation function (say, based on some key in the record).

The analysis carried out in this work showed that changing the parameter of the packet size (batch.size) can increase the bandwidth, reduce the load when processing packets, as well as the operations of streaming I/O. And at low load, Kafka can increase the delay in sending packages, as the producer will expect the batch to be ready.

Based on the results obtained, it can also be argued that it is not enough just to read and write a data stream, the main goal is to ensure the processing of streams in real time. For this purpose, it can be used the Kafka Streams API, a streaming processor that receives continuous data streams from input channels, performs some processing on that input, and provides continuous data output.

Thus, thanks to the new architecture, it was possible to implement the possibility of scaling without increasing the number of equipment. At the same time, the system was transferred to a new principle of operation - modularity (because Kafka allows you to connect modules and applications regardless of the programming language), and to ensure further escalation of the client's business.

Keywords: business analysis, architectural solutions, Apache Kafka, Apache Connector, Apache Streams, horizontal scaling, distributed computing, parallel computing.

Стаття: надійшла до редакції 06.04.2020,
доопрацьована 20.04.2020,
прийнята до друку 22.04.2020.