COMPUTER SCIENCE

WEB MINING НА ОСНОВІ ЛЯМБДА АРХІТЕКТУРИ

Фінчук О. В., *associate professor*

Україна, м. Київ, Національний технічний університет України «Київський політехнічний інститут ім. І. Сікорського», ФІОТ, кафедра ОТ, студент

DOI: https://doi.org/10.31435/rsglobal_ws/12072018/6004

ARTICLE INFO

Received: 13 May 2018
Accepted: 28 June 2018
Published: 12 July 2018

KEYWORDS

web mining;
web scraping;
lambda architecture;
big data;
batch processing;
stream processing;
cluster, apache spark;
apache kafka.

ABSTRACT

Amount of data in the web is growing rapidly from day to day and for normal human it's hard to realize what is actually happening in the world at the current moment, so it is required to have a system which allows collect and analyze huge amount of variety data from different web resources. The system should be able to process a lot of data fast and be capable to store all collected data for future processing. How to achieve this with use of lambda architecture is described in this article.

Citation: Фінчук О. В. (2018) Web Mining na Osnovi Liambda Arkhitektury. *World Science.* 7(35), Vol.2. doi: 10.31435/rsglobal_ws/12072018/6004

За останні декілька років розмір даних на електронних носіях виріс в десятки разів і продовжує зростати експоненційно. Спостерігати за цим процесом та проводити аналітику стає дедалі важче. Проте всі зацікавлені у детальному аналізі процесів, що відбуваються у світі. Наприклад, тенденції в інтересах людей, популярність певних технологій, зміни прихильності людей до розваг. Для отримання таких даних необхідно мати систему, яка б змогла збирати досить багато даних із необхідних ресурсів та аналізувати їх відповідно до поставлених вимог.

Для початку опишемо вимоги до цієї системи:

- Актуальність даних повинна бути дуже високою, оскільки важливо розуміти, що відбувається прямо зараз, тобто затримка між оновленням зовнішньої системи та появою нових даних у нашій системі повинна бути мінімальною.
- Можливість архівування даних для подальшої обробки, або зміни існуючої. Це необхідно, щоб завжди можна було добавити нові статистичні відображення на уже зібрані дані, без необхідності повторного збору. Також це дозволить виправити помилки в існуючих алгоритмах.
- Здатність до лінійного масштабування. Система повинна справлятися з дуже великим об’ємом даних, і для досягнення цього повинно бути достатньо просто добавити більше машин. Проте при невеликому навантажені повинно бути достатньо використання однієї чи декількох машин.
- Відмово стійкість. При поломці якогось одного компонента система бути справною.

Для досягнення поставлених вимог чудово підходить **лямбда архітектура.**

У світі великих даних існує два основних підходи до обробки. Перший – це регулярна обробка великого об’єму накопичених даних, що зазвичай займає багато часу, інколи до

декількох годин. Другий же це обробка мікро-пакетів даних, що поступають постійно з частотою від декількох мілісекунд.

Лямда архітектура поєднує в собі переваги пакетної та потокової обробки даних. Це поєднання відбувається за рахунок інкрементуючих алгоритмів. Таких підхід намагається збалансувати пропускну спроможність, мінімальну затримку та відмовостійкість. При цьому використовується швидкісна обробка в режимі близькому до реального часу. З іншого ж боку тут також присутня пакетна обробка, щоб виключити попередні помилки програмістів, а також для того, щоб добавляти нові обчислення на існуючі сирі дані.

Лямбда архітектура орієнтована на постійно вхідні дані із якихось ресурсів, тому вона ідеально підходить для web mining. Розглянемо її детальніше.

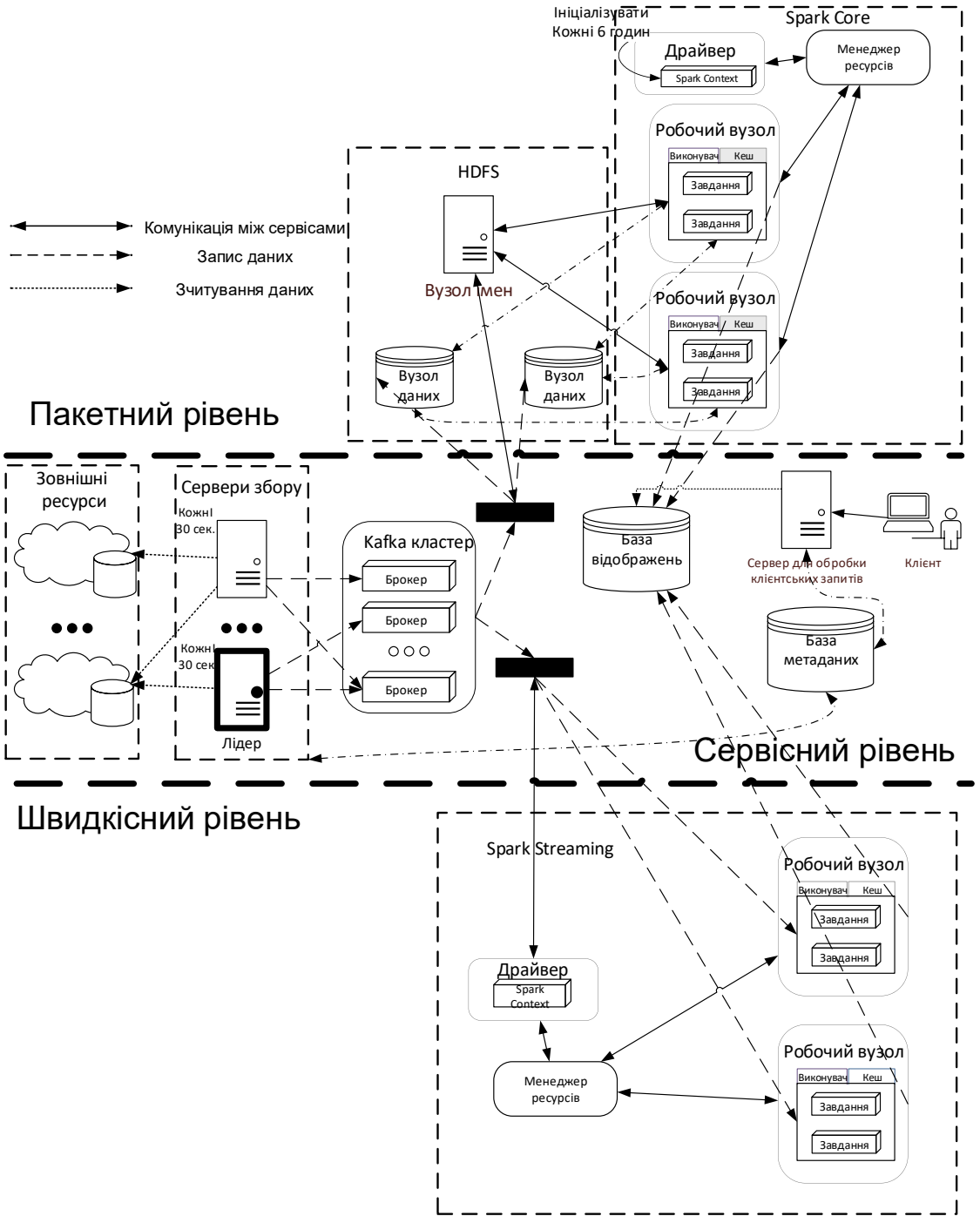

Рис. 1 – лямбда архітектура для web mining

Як видно з рисунку 1 архітектура складається з трьох основних рівнів:

- Пакетний рівень – сховище усіх раніше зібраних сирих даний, які накопичуються у так зване “озеро даних”. Через певні інтервали часу проводиться повне переобчислення відображень із цілого, або частини, озера даних. Варто вказати, що на цьому рівні архівні дані нікуди не діваються, а просто постійно накопичуються. Цей рівень існує для можливості додавання нових відображень та виправлення помилок в існуючих.
- Сервісний рівень – об’єднання результатів пакетного і швидкісного рівнів. На даному рівні відбувається індексація обчислених відображень із пакетного рівня. Оскільки обчислення може тривати годинами, то результат уже буде застарілий. Цю різницю покривають результати швидкісного рівня, оскільки затримка на цьому рівні всього декілька секунд. Також на цьому рівні відбувається обробка усіх клієнтських запитів, тобто є вхідною точкою для користувачів.
- Швидкісний рівень – компенсація затримки на пакетному рівні. На даному рівні проходить обробка мікро-пакетів, що містять в собі найновіші дані. Далі обчислені відображення об’єднуються з уже існуючими, тому є можливість бачити складну статистику на великих даних у режимі близькому до реального часу.

Проте на ці рівні повинні постійно поступати нові дані. За це відповідає модуль збору даних, який регулярно посилає запити на зовнішні ресурси. Деякі ресурси підтримують машинозчитувальний формат та програмний інтерфейс (Web API), проте інколи цікавить інформація, яку можна отримати із HTML сторінок. Цей процес називається **web scraping**. Програма імітує запити людини і далі інтерпретує отримані веб-сторінки і структуровану інформацію для подальшої обробки.

При необхідності збору з багатьох ресурсів необхідно цей процес виконувати паралельно на декількох машинах. Для цього перед початком роботи обирається лідер, який відповідає за сфери збору. Уся метаінформація про джерела збору, цікаві нам дані, дату останнього оновлення береться із бази метаданих, яка оновлюється на сервісному рівні користувачами. Після цього усі дані направляються на пакетний та швидкісний рівень.

Весь процес збору та побудови статистичних відображень являє собою **web mining**. Складається він із чотирьох етапів:

1. Вхідний етап – збір сирих даних
2. Етап обробки – перетворення у необхідну структуру.
3. Етап моделювання – побудова відображень
4. Етап аналізу – інтерпретація отриманих результатів.

При реалізації системи також необхідно обрати правильні технології для досягнення кращої ефективності. Із модуля збору дані повинні постійно надходити на пакетний і швидкісний рівень. Для цього підходить Apache Kafka – розподілена система посилки повідомлень, яка забезпечує гарантовану доставку із мінімальною затримкою.

Після того як дані потрапляють на пакетний рівень їх краще зберігати на HDFS (Hadoop distributed file system) – розподілена файлова система, яка може зберігати до 200 петабайт даних. Важливо використовувати розподілену файлову систему не лише через розмір, який може зберігатись, а і через можливість паралельного доступу із різних машин, що дуже важливо для обробки великих даних. Обробка ж сама відбувається з використанням apache spark, оскільки він має високі показники ефективності та автоматичні оптимізації за рахунок максимального використання оперативної пам’яті.

Для реалізації швидкісного рівня необхідна висока пропускна здатність, мінімальна затримка та можливість обчислення за останній змінний проміжок часу. Цим вимогам відповідає Apache Spark Streaming. Висока пропускна спроможність досягається за рахунок розміщення на декількох машинах, побудові ефективного плану виконання та внутрішніх оптимізацій.

Після обробки на швидкісному рівні дані потрапляють у сервісний рівень, де за допомогою інкрементуючих алгоритмів додаються до існуючих відображень та зберігаються там. По-перше база даних повинна підтримувати швидкі випадкові зчитування та запис. По-друге, структури відображень динамічні і дуже різні. По-третє, пошук повинен проводитись дуже швидко, через це в даному випадку потрібно використовувати NoSQL базу даних, а саме ElasticSearch. Він дозволяє задавати шаблон документів і на їх основі будує обернений індекс, що значно пришвидшує пошук.

Для проведення тестування та порівняння ефективності використовувався тестовий сервер, який імітував роботу зовнішніх ресурсів. Для порівняння було обрано такі умови:

- Простий збір на одній машині.

- Збір на повністю налаштованому кластері із рівнем паралельності 2
- Збір на повністю налаштованому кластері із рівнем паралельності 4

Рис. 2 – Тестування системи

На рисунку 2 зображено результати тестування системи. Опишемо як проходило тестування. Вимірюється час між отриманням запитів із мок-сервера та зберіганням їх у базі. Для проведення таких тестів виконуються такі кроки:

1. Запускається мок-сервер на певному порту
2. Для мок-серверу задаються вхідні дані, які будуть віддаватися на вхідні запити. Ці дані будуть віддаватись тільки після сигналу про старт тестування.
3. Запускають тестову базу даних, у якій будуть зберігатись результати
4. В залежності від умови тестування запускаються сервіси на одній машині, або на кластері із тестовими конфігураціями.
5. Незалежна система тестування посилає запит про старт тестування, починає замір часу та регулярно перевіряє стан обробки. Замір часу зупиняється, коли досягається певний стан.

Проведемо аналіз отриманих результатів для кожної системи.

- Перша система має кращі показники до 7000 записів, проте далі вона справляється значно гірше. На графіку відсутнє значення часу для 100000 записів, оскільки система не витримала навантаження і виникла помилка про завершення пам'яті. Також на графіку видно різкий перелом після 5000 записів, що свідчить про 100% після цього моменту.
- Друга система на початку показує себе гірше, оскільки присутні витрати на комунікацію та пересилання між сервісами. Система повністю справилась із поставленими вимогами, проте після 50000 записів можна спостерігати повне навантаження на сервери.
- У третьому випадку система з легкістю справилася з поставленим навантаження, проте було затрачено ще більше часу на пересилку через більшу кількість машин.

Отже, в даній статті була описана лямбда архітектура та її реалізація для збору та аналізу даних із різноманітних зовнішніх ресурсів. При такому підході абсолютно різна статистика із дуже великого набору даних може відображатися у режимі наближеному до реального часу. Це дозволить проводити складний аналіз процесів дуже швидко. В подальших

роботах планується дослідження ефективного аналізу зібраних даних а також побудова правильних моделей.

ЛІТЕРАТУРА

1. Big Data: Principles and best practices of scalable realtime data systems 1st Edition, 2015. -328с.
2. Patterns of Enterprise Application Architecture 1st Edition, 2002. – 560с.
3. lambda-architecture-with-apache-spark [Електронне джерело] – режим доступу: https://speakerdeck.com/mhausenblas/lambda-architecture-with-apache-spark?slide=1
4. A repository dedicated to the Lambda Architecture [Електронне джерело] – режим доступу: http://lambda-architecture.net/
5. High Performance Spark Best Practices for Scaling and Optimizing Apache Spark, 2017-175с.
6. Hadoop: The Definitive Guide, 4th Edition, 2015. -235с.
7. Testing with JMeter [Електронне джерело] – режим доступу: https://jmeter.apache.org/