Мікросервіс з обробки великих даних

Корнійчук Роман Сергійович, студент групи КА-54 ІПСА

Науковий керівник:

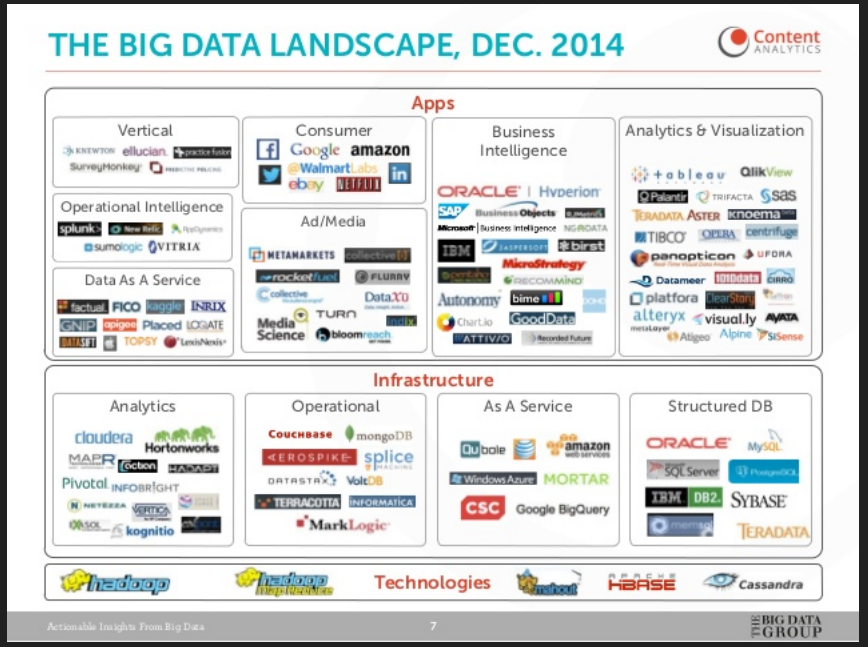
Тимощук О.Л.,

к.т.н., доцент кафедри математичних методів системного аналізу ІПСА НТУУ «КПІ ім. Ігоря Сікорського»

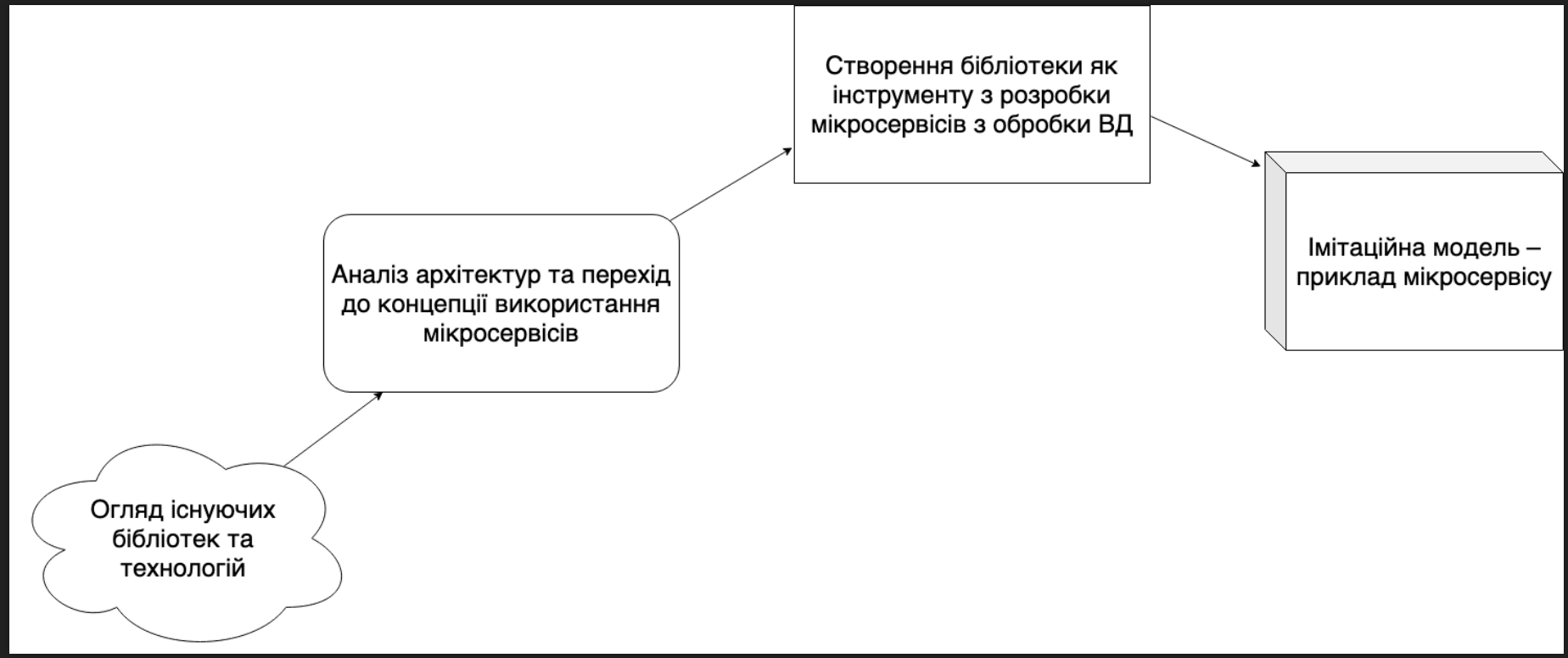
Актуальність

- Після ери Big Data дані будуть ще більшими, вороття немає
- Доступність великих даних як методології зростає не так швидко, як потреба у них
- Великі дані доступні кожному

BIG DATA & AI LANDSCAPE 2018


THE BIG DATA LANDSCAPE, DEC. 2014
Content ANALYTICS
Apps
Vertical
Consumer
Business Intelligence
Analytics & Visualization
Operational Intelligence
Ad/Media
Data As A Service
Infrastructure
Analytics
Operational
As A Service
Structured DB
Technologies
Actionable Insights From Big Data
7
THE BIG DATA GROUP

Маршрутна карта роботи



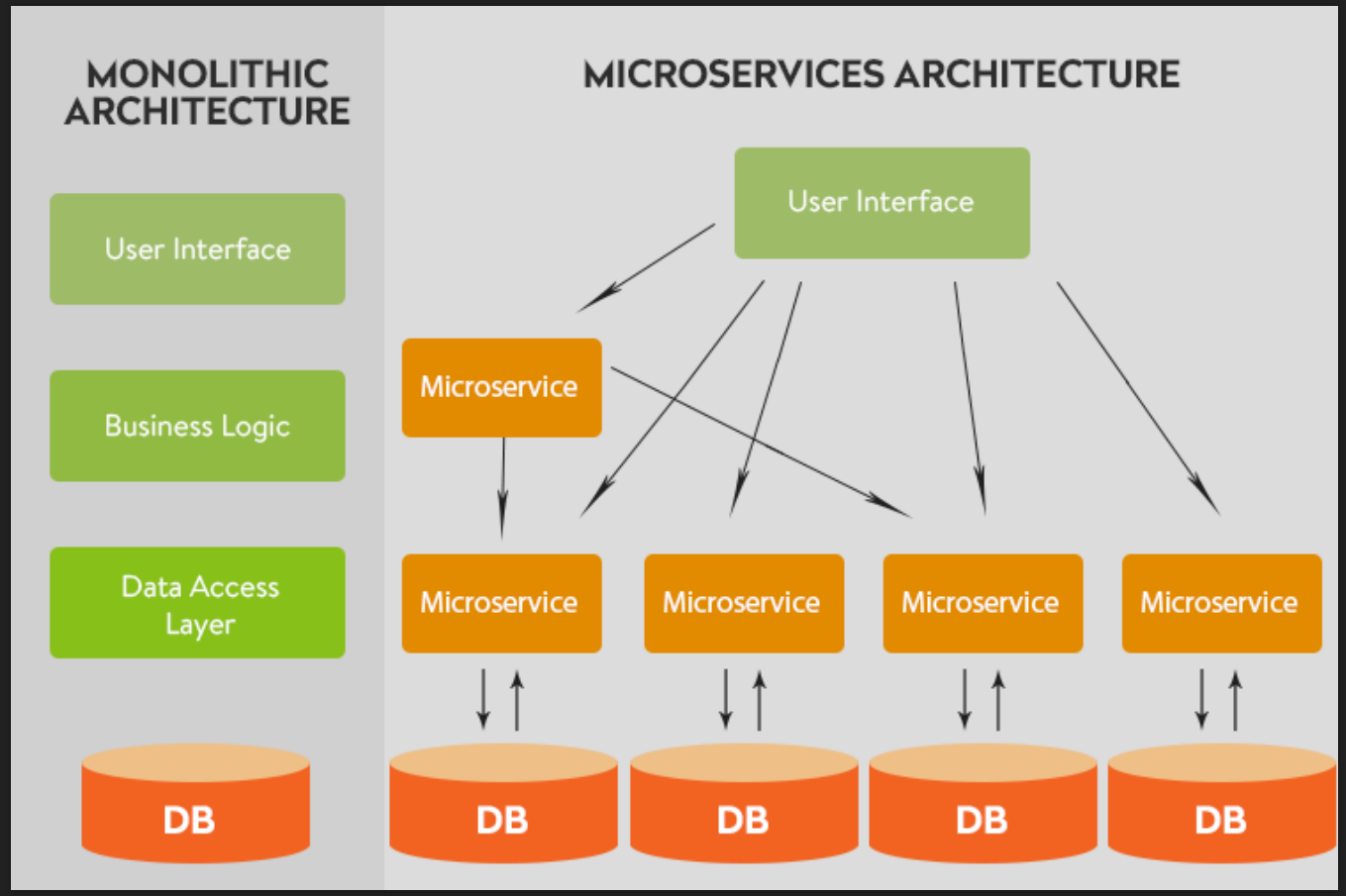
Проблеми розробки ПЗ з обробки великих даних

- Розробка навіть простих рішень вимагає глибокого вивчення існуючих технологій
- Невигідно велика на початку впровадження власних розробок множина альтернатив

Постановка задачі

- Спростити розробку простих програмних засобів для обробки великих даних
- Використати для цього сучасні технології
- Обмежити розробника у мінімально необхідному наборі можливостей для прискорення розробки простих програмних засобів

Мікросервісна архітектура



Основні складові простих мікросервісів

- REST API для отримання повідомлення про запуск виконання завдання, що містить всі його деталі
- Черга завдань
- Мінімально налаштоване середовище виконання на боці кластера Apache Spark
- Налаштовувані callback-функцій для обробки результату виконання завдань (наприклад, надсилання його іншому мікросервісу)

Приклад використання

- Кластеризація локальних файлів методом locality-sensitive hashing (LSH)
- Callback-функція записує результат у файл
- Подальше розширення API дозволить завантажувати та вивантажувати ці файли

Алгоритми кластеризації

- Алгоритм k-середніх
- Алгоритм HCM
- *Алгоритм нечіткої кластеризації Fuzzy C-means*
- Алгоритм FRiS-Cluster
- Алгоритм FRiS-Stolp
- Алгоритм LSH (Locality-sensitive hashing)

LSH

An *LSH family*[1][4][5] $\mathcal{F}$ is defined for a metric space $\mathcal{M} = (M, d)$, a threshold $R > 0$ and an approximation factor $c > 1$. This family $\mathcal{F}$ is a family of functions $h : \mathcal{M} \to S$ which map elements from the metric space to a bucket $s \in S$. The LSH family satisfies the following conditions for any two points $p, q \in \mathcal{M}$, using a function $h \in \mathcal{F}$ which is chosen uniformly at random:

- if $d(p, q) \leq R$, then $h(p) = h(q)$ (i.e.,p and q collide) with probability at least P_1,
- if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most P_2.

A family is interesting when $P_1 > P_2$. Such a family $\mathcal{F}$ is called (R, cR, P_1, P_2)-*sensitive*.

Alternatively[6] it is defined with respect to a universe of items U that have a similarity function $\phi : U \times U \to [0, 1]$. An LSH scheme is a family of hash functions H coupled with a probability distribution D over the functions such that a function $h \in H$ chosen according to D satisfies the property that $Pr_{h \in H}[h(a) = h(b)] = \phi(a, b)$ for any $a, b \in U$.

Locality-preserving hashing [edit]

A **locality-preserving hashing** is a hash function *f* that maps a point or points in a multidimensional coordinate space to a scalar value, such that if we have three points *A*, *B* and *C* such that

$$|A - B| < |B - C| \Rightarrow |f(A) - f(B)| < |f(B) - f(C)|.$$

In other words, these are hash functions where the relative distance between the input values is preserved in the relative distance between the output hash values; input values that are closer to each other will produce output hash values that are closer to each other.

This is in contrast to cryptographic hash functions and checksums, which are designed to have random output difference between adjacent inputs.

Locality preserving hashes are related to space-filling curves.

Amplification [edit]

Given a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$, we can construct new families $\mathcal{G}$ by either the AND-construction or OR-construction of $\mathcal{F}$.[1]

To create an AND-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if all $h_i(x) = h_i(y)$ for $i = 1, 2, \ldots, k$. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a (d_1, d_2, p_1^k, p_2^k)-sensitive family.

To create an OR-construction, we define a new family $\mathcal{G}$ of hash functions g, where each function g is constructed from k random functions $h_1, \ldots, h_k$ from $\mathcal{F}$. We then say that for a hash function $g \in \mathcal{G}$, $g(x) = g(y)$ if and only if $h_i(x) = h_i(y)$ for one or more values of i. Since the members of $\mathcal{F}$ are independently chosen for any $g \in \mathcal{G}$, $\mathcal{G}$ is a $(d_1, d_2, 1 - (1 - p_1)^k, 1 - (1 - p_2)^k)$-sensitive family.

Аналіз отриманих результатів

- У рамках даної роботи було розроблено бібліотеку для розробки мікросервісів з обробки великих даних, а також мікросервіс із кластеризації векторів цілих чисел як приклад її застосування, було надано інструкції з використання цієї бібліотеки.
- Приведений приклад її застосування може бути застосовано як повноцінний мікросервіс, якщо реалізувати його DevOps частину (впровадження) і розгорнути у місці його подальшого прикладного використання, а також замінити орієнтованість на тестові дані обробкою реальних даних.

Дякую за увагу!