Temat: Porównanie języka Java i Python w użyciu baz SQL i NoSQL.

Anna Żurczak 127227

Ocena efektywności systemów komputerowych – laboratoria

Projekt I

1. Wstęp

Zadanie dotyczyło porównania języków Java i Python w odniesieniu do obsługi baz SQL i NoSQL, a właściwie bibliotek które pozwalają na wykonywanie operacji na tych bazach. Jako baza SQL zostało wykorzystane SQLite, a jako baza NoSQL użyto MongoDB. Celem zadania było sprawdzenie, która para narzędzi najbardziej efektywnie – w najkrótszym czasie przy najniższym stopniu wykorzystania zasobów - wykona załadowanie bazy, danymi z pliku tekstowego.

1. Metodyka badania

Pomiary zostały wykonane dla 4 par : Java + SQLite, Python + SQLite, Java + MongoDB, Python + MongoDB.

Dane do załadowania bazy pochodzą z <http://millionsongdataset.com/>. Pomiary wykonano dla 5 rozmiarów plików – 50MB, 100MB, 150MB, 250MB, 500MB. Każdy plik ma następującą strukturę: user\_id<SEP>song\_id<SEP>date(timestamp).

W SQL stworzono dwie tabele: Samples (user\_id, song\_id, foreign key) i Dates(id, day, month, year). Plik czytano linia po linii, każdą krotkę wstawiano pojedynczo.

W MongoDB stworzono dwie kolekcje Samples (\_id, user\_id, song\_id, key to date) i dates (\_id, id, day, month, year). Plik czytano linia po linii. W Pythonie tworzony był json zawierający 1000 linii, a w Javie kolekcja zawierająca 1000 obiektów BasicDBObject.

Biblioteki użyte dla Javy : mongo-java-driver w wersji 3.9.1, sqlite-jdbc w wersji 3.30.1

Biblioteki użyte dla Pythona : pymongo w wersji 3.9.0 , sqlite3 w wersji 2.6.0

Użyte bazy danych : MongoDB w wersji 4.2.2 i SQLite w wersji 3.30.1

Parametry komputera na którym wykonano pomiary :

|  |  |
| --- | --- |
| System operacyjny | Windows 10 |
| Procesor | Intel Core i5-7300HQ (4 rdzenie) |
| RAM | 8.00 GB |

Do pomiarów w pythonie użyto bibliotek: time, psutil i os, a z nich funkcji:

current\_process = psutil.Process(os.getpid())

current\_process.cpu\_percent()

current\_process.memory\_info()[0] # [0] – rss(resident set size)

t\_start = time.time()

Do pomiarów w Javie użyto klas OperatingSystemMXBean z pakietu com.sun.management, Runtime i System z java.lang.

long startTime = System.*currentTimeMillis*();

(com.sun.management.OperatingSystemMXBean) bean).getProcessCpuLoad()

Runtime.*getRuntime*().totalMemory() - Runtime.*getRuntime*().freeMemory()

1. Wyniki

Jako miarę tendencji centralnej wybrano średnią, ponieważ suma wartości pomiarów ma sens i ich rozkład nie jest skośny. Jako miarę rozrzutu wybrano odchylenie standardowe – zostanie zaznaczone w postaci słupków błędu na wykresie poniżej.

* 1. Java

1. SQLite

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar pliku [MB] | Procent wykorzystania procesora[%] | Pamięć [MB] | Czas[s] |
| 50 | 13,76 | 69,68 | 21,44 |
| 100 | 19,68 | 80,49 | 7,26 |
| 150 | 17,90 | 93,23 | 12,95 |
| 250 | 19,05 | 66,11 | 19,66 |
| 500 | 17,57 | 81,81 | 46,46 |

Tabela Java + SQLite - średnia

1. MongoDB

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar pliku [MB] | Procent wykorzystania procesora[%] | Pamięć [MB] | Czas[s] |
| 50 | 8,96 | 60,55 | 15,20 |
| 100 | 7,15 | 63,12 | 31,00 |
| 150 | 7,81 | 52,46 | 45,40 |
| 250 | 6,85 | 59,52 | 77,40 |
| 500 | 6,89 | 64,81 | 155,40 |

Tabela Java + MongoDB - średnia

* 1. Python

1. SQLite

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar pliku [MB] | Procent wykorzystania procesora[%] | Pamięć [MB] | Czas[s] |
| 50 | 24,16 | 17,03 | 27,76 |
| 100 | 24,16 | 17,03 | 36,42 |
| 150 | 24,16 | 17,03 | 53,98 |
| 250 | 24,16 | 17,03 | 92,91 |
| 500 | 24,16 | 17,03 | 181,01 |

Tabela Python + SQLite - średnia

1. MongoDB

|  |  |  |  |
| --- | --- | --- | --- |
| Rozmiar pliku [MB] | Procent wykorzystania procesora[%] | Pamięć [MB] | Czas[s] |
| 50 | 17,03 | 19,72 | 38,78 |
| 100 | 17,03 | 19,72 | 72,97 |
| 150 | 17,03 | 19,72 | 110,95 |
| 250 | 17,03 | 19,72 | 181,66 |
| 500 | 17,03 | 19,72 | 364,47 |

Tabela Python + MongoDB - średnia

Wykres Średnie wykorzystanie procesora

Wykres Średni rozmiar wykorzystywanej pamięci

Wykres Średni czas wykonywania

1. Podsumowanie

Najbardziej efektywną parą okazała się para Java + MongoDB, wykorzystuje dużo więcej pamięci niż Python, jednakże robi to w krótkim czasie i przy dziesięcioprocentowym użyciu procesora.

Trzeba zwrócić uwagę na różnicę w pobieraniu danych, w javie wartość wykorzystywanej pamięci w danej chwili to cała pamięć jaką wykorzystuje maszyna javy, a w pythonie to pamięć która zajmują zmienne i program – stąd linia prosta na wykresie 2.

Java jest szybsza, ponieważ jest językiem kompilowanym, Python jest językiem interpretowalnym, który określa rodzaj danych w czasie wykonania co czyni go wolniejszym. Połączenie tych języków z SQLite okazało się szybsze niż z MongoDB [Wykres 3].

Dla każdego przypadku zostało wykonane pięć pomiarów, a z nich obliczono odchylenie standardowe. Słupki błędu najbardziej widoczne są na wykresie 2, czyli średnim wykorzystaniu pamięci dla javy. Dla czasu [Wykres 3] odchylenie wynosi praktycznie 0, a dla wykorzystania procesora [Wykres 1] największy rozrzut wystąpił dla SQLite.

Warto również porównać dla którego języka napisanie testów okazało się prostsze, można jednoznacznie stwierdzić, że w przypadku bazy SQLite i MongoDB biblioteki są prostsze w użyciu dla Pythona.