Wyszukiwanie w Riak

• Prosty model danych oznacza brak zbiorów, transakcji czy złączeń (bo nie ma kolumn ani wierszy).

Dane można przeszukiwać poprzez:

• riak search: rozproszony silnik wyszukiwania pełnotekstowego,

• secondary indexing (2i): możliwość przypisywania do obiektów tagów, które potem mogą stanowić element zapytania,

• MapReduce: deﬁniowanie procedur przy pomocy JavaScript lub Erlanga.

Riak Search

Stosujemy wtedy, gdy:

• zbieramy i parsujemy treści takie jak artykuły, blogi, itd.,

• chcemy indeksować dane JSON,

• potrzebujemy szybkiego dostępu i zarazem bogatych

możliwości odpytywania.

• Nie stosujemy, gdy wystarczyłoby tagowanie danych, dla

danych binarnych lub gdy wymagamy pewnej spójności

(wyszukiwarka nie wspiera *read repair*).

Secondary Indexes (2i)

• Udostępniają możliwość tagowania obiektów.

• Dostępne typy: liczby całkowite i ciągi znaków

(również binaria).

• Dokładne dopasowanie lub zakres wartości.

• Możliwa paginacja, streaming.

• Również można wykorzystać jako wejście do MR.

Używamy, gdy:

• chcemy szukać w oparciu o coś innego niż para wiadro/klucz,

• przechowujemy bloby i chcemy mieć możliwość sprawnego

wyszukiwania.

• Nie używamy, gdy:

• mamy >512 partycji,

• potrzebujemy czegoś innego niż dokładne dopasowanie lub

zakresy,

• wymagamy złożonych zapytań.

MapReduce

Faza *map* wykonywana na węzłach posiadających określone dane,

*reduce* – na węźle któremu zlecono zadanie.

Używamy MapReduce:

• gdy znamy zbiór par wiadro/klucz do przetworzenia,

• gdy wyniki mają zawierać wartości bądź ich fragmenty (a nie

same klucze),

• gdy wymagana jest większa elastyczność niż ta w wyszukiwarce

czy 2i.

• Nie używamy do przeglądu całego wiadra, lub gdy chcemy

deterministycznego czasu wyszukiwania.



Model danych Riak

• Dane (klucz+wartość) – nazywane obiektami –

pogrupowane w tzw. wiadrach (buckets).

• Pary wiadro+klucz są hashowane SHA1 – powstają

skróty o długości 160 bitów.

• Przestrzeń kluczy jest dzielona na partycje

(domyślnie 64) – każda partycja odpowiada za

zakres zawierający 2^160/n wartości.





Przykładowy kod użytkowania:

Start riaka:

$ riak start

Odpalamy erlanga

$ erl -name riaktest@127.0.0.1 -setcookie riak

Sprawdzamy czy serwer odpowiada:

(riaktest@127.0.0.1)1> RiakNode = 'riak@127.0.0.1'.

(riaktest@127.0.0.1)2> net\_adm:ping(RiakNode).

Łączymy klienta do riaka

(riaktest@127.0.0.1)2> {ok, C} = riak:client\_connect(RiakNode).

Przykładowe dodawanie danych do bazy

(riaktest@127.0.0.1)3> O0 = riak\_object:new(<<"groceries">>, <<"mine">>, ["bread"]).

(riaktest@127.0.0.1)4> C:put(O0, 1).

Możemy odczytać co jest w bazie:

(riaktest@127.0.0.1)5> {ok, O1} = C:get(<<"groceries">>, <<"mine">>, 1).

Dodawanie kolejnych wartości do riaka

(riaktest@127.0.0.1)6> O2 = riak\_object:update\_value(O1, ["milk" | V]).

(riaktest@127.0.0.1)7> C:put(O2, 1).

Sprwdzamy dostępne key

(riaktest@127.0.0.1)8> C:list\_keys(<<"groceries">>).