Проектна задача по предметот Неструктутрирани бази на податоци

Компаративна анализа на Key-Value NoSQL бази на податоци:

Имплементација и перформанси со Redis и Riak

Членови:

Мартин Митев 216040

Мила Велкова 211055

Марија Димитреиска 211117

Иван Пупиноски 223260

Ментор:

д-р Слободан Калајџиски

Скопје, Јуни 2025

Содржина

Вовед 3
Методологија 4
Инсталација и конфигурација 4
Инсталација и конфигурација на Redis 4
Инсталација и конфигурација на Riak 5
Анализа и моделирање на податоците 6
Начини на агрегација на податоците 6
Агрегација во Redis 6
Агрегација во Riak 7
Дефинирање на сценарија за пристап до податоци (Прашалници) 7
Едноставни прашалници (Филтрирања) 8
Сложени прашалници (Спојување на податоци од повеќе инстанци) 8
Доста сложени прашалници (Агрегирани извештаи) 8
Добиени резултати 8
Резултати од спроведените експерименти 8
Табела 1: Време на извршување на едноставни прашалници 8
Табела 2: Време на извршување на сложени прашалници 9
Табела 3: Време на извршување на агрегирани прашалници 9
Дискусија за добиените резултати 9
Визуелизација 10
Слика 1: Споредба на време на извршување на прашалници (Redis vs Riak) 10
Слика 2: Споредба на време на извршување на прашалници по категорија 11
Заклучок 11
Користена литература 12

Вовед

Во денешно време, со експоненцијалниот раст на податоците генерирани од најразлични извори, традиционалните релациони бази на податоци честопати наидуваат на ограничувања во однос на скалабилноста, перформансите и флексибилноста. Ова доведе до појава и широка примена на NoSQL (Not only SQL) базите на податоци, кои нудат алтернативни модели за складирање и управување со податоци, прилагодени за справување со големи количини на неструктурирани и полуструктурирани податоци.

Предмет на оваа проектна задача е детална анализа и компарација на Key-Value NoSQL базите на податоци, кои претставуваат еден од основните модели во NoSQL екосистемот. Посебен фокус е ставен на практичната имплементација и анализа на перформансите на две конкретни технологии од овој тип: Redis и Riak. Преку оваа проектна задача ја истраживме нивната способност за импортирање, моделирање и агрегација на дадени податочни множества, како и нивната ефикасност при извршување на различни типови на прашалници – од едноставни филтрирања до сложени агрегирани извештаи.

При решавањето на поставениот проблем, користевме релевантна стручна литература и официјална документација за NoSQL базите на податоци, со посебен акцент на Redis и Riak. Детални информации за инсталација, конфигурирање и работа со овие бази се црпени од нивните официјални веб-страници и придружни ресурси. Овој преглед на литература ни овозможи длабинско разбирање на проблематиката и систематски пристап кон изнаоѓање на оптимални решенија.

Методологија

Инсталација и конфигурација

Овој дел детално ја објаснува постапката за добивање на решението за поставениот проблем, фокусирајќи се на дизајнот, имплементацијата и користењето на Key-Value NoSQL базите на податоци. Како дел од задачата, тимот е поделен на два под-тима, при што секој работи со различна база на податоци од типот Key-Value: Redis и Riak, на исто податочно множество во кое централниот ентитет е филмот.

Инсталација и конфигурација на Redis

Инсталацијата на Redis е извршена локално, со искористување на **Docker Desktop**, што овозможува брза и изолирана околина за работа. Командата што е искористена за подигање на Redis инстанца е следнава:

docker run --name redis -p 6379:6379 -d redis

Оваа команда стартува Docker контејнер со име `redis`, мапирајќи ја стандардната порта на Redis (6379) од контејнерот на истата порта на локалниот хост. `-d` аргументот го стартува контејнерот во позадина (detached mode).

Поврзувањето со Redis базата на податоци од Python апликацијата се реализира преку `redis-py` библиотеката, поврзувајќи се на `localhost` и порта `6379`, што се стандардни вредности.

```
import pandas as pd
import redis
import json

# Вчитување на податоци од CSV датотека
# Поради големината на оригиналното податочно множество (1M записи), за
целите на тестирање и развој, се користат првите 5000 записи.
df = pd.read_csv("IMDB TMDB Movie Metadata Big Dataset (1M).csv",
low_memory=False, nrows=5000)

# Отстранување на редови каде 'id' или 'title' недостасуваат, бидејќи се
критични за клучевите.
df = df.dropna(subset=["id", "title"])

# Воспоставување конекција со Redis базата на податоци
redis_db = redis.Redis(host='localhost', port=6379, db=0)

# Итерирање низ DataFrame и вчитување на податоците во Redis
# Секој филм се зачувува како Key-Value пар, каде клучот е 'movie:ID', а
вредноста е JSON репрезентација на филмскиот објект.
for _, row in df.iterrows():
    key = f"movie:{int(row['id'])}"
    value = json.dumps(row.dropna().to_dict())
    redis_db.set(key, value)
```

```
print("□ Data loaded into Redis.")
```

Ова ја демонстрира постапката на вчитување на првите 5000 филмски записи од даденото CSV податочно множество во Redis. Секој филм се складира како JSON стринг под клуч во формат movie:ID.

Инсталација и конфигурација на Riak

Инсталацијата на Riak KV е исто така извршена локално преку **Docker Desktop**, следејќи ги препорачаните упатства за подигање на самостојна инстанца. Командата за стартување на Riak контејнерот е следнава:

docker run --name=riak -d -p 8087:8087 -p 8098:8098 basho/riak-kv

- ***--name=riak:*** Доделува име на контејнерот "riak" за полесно управување.
- ***-d:*** Го стартува контејнерот во позадина.
- ***-p 8087:8087:*** Ја мапира Riak HTTP API портата (8087) од контејнерот на портата 8087 на локалниот хост. Оваа порта се користи за комуникација со Riak преку HTTP клиенти.
- ***-p 8098:8098:*** Ја мапира Riak Protocol Buffers (PBC) портата (8098) од контејнерот на портата 8098 на локалниот хост. Оваа порта се користи од страна на Riak клиенти (вклучувајќи ги и повеќето официјални клиентски библиотеки) за ефикасна бинарна комуникација.
- ***basho/riak-kv:*** Одредува Docker Image кој треба да се повлече и стартува, во овој случај официјалниот image за Riak Key/Value базата на податоци.

Поврзувањето со Riak базата на податоци од Python апликацијата се реализира преку `riak-python-client` библиотеката. Важно е да се напомене дека стандардната порта за Protocol Buffers (PBC) во Riak е 8087, што се користи во кодот за поврзување (`pb_port=8087`).

```
import pandas as pd
import json
import riak

# Вчитување на податоци од CSV датотека
# Како и за Redis, и овде се користат првите 5000 записи од податочното
множество за целите на развој, тестирање и споредба.
df = pd.read_csv("IMDB TMDB Movie Metadata Big Dataset (1M).csv",
low_memory=False, nrows=5000)

# Отстранување на редови каде 'id' или 'title' недостасуваат, бидејќи се
критични за клучевите.
df = df.dropna(subset=["id", "title"])

# Воспоставување конекција со Riak базата на податоци
# Се користи Protocol Buffers (PBC) протоколот за конекција на порта 8087.
riak_db = riak.RiakClient(pb_port=8087, protocol='pbc')

# Дефинирање на "bucket" (аналог на колекција/табела во други NoSQL бази)
```

```
# Сите филмски објекти ќе бидат зачувани во овој bucket.
bucket = riak_db.bucket("movies")

# Итерирање низ DataFrame и вчитување на податоците во Riak
# Секој филм се зачувува како Riak објект, каде клучот е 'movie:ID', а
вредноста е JSON репрезентација на филмскиот објект.
for _, row in df.iterrows():
    key = f"movie:{int(row['id'])}"
    value = json.dumps(row.dropna().to_dict())

    # Креирање нов Riak објект со зададениот клуч и податоци
    obj = bucket.new(key, data=value)

    # Зачувување на објектот во Riak
    obj.store()

print("□ Data loaded into Riak.")
```

Ова ја прикажува постапката на вчитување на првите 5000 филмски записи од даденото CSV податочно множество во Riak. Секој филм се складира како JSON стринг во `movies` bucket-от, под клуч во формат `movie:ID`.

Анализа и моделирање на податоците

За реализација на проектната задача е користено податочно множество **IMDB & TMDB Movie Metadata Big Dataset (over 1M)**. Пред импортирањето, извршена е детална анализа на структурата на податоците и меѓусебните врски, со цел подобро разбирање на доменот на филмови и поврзани метаподатоци. Централниот ентитет е филмот, кој содржи атрибути како ID, наслов, година на издавање, просечна оцена, главен актер и листа на жанрови кои се едни од поглавните додека пак податочното множество има и други атрибути како јазик, буџет, популарност, статус итн.

Врз основа на оваа анализа, дизајниран е соодветен модел на податоци за Key-Value базите. Примарниот модел е *директно складирање на филмските објекти* како JSON стрингови, каде `movie:ID` е клуч, а целиот филмски објект е вредност.

Начини на агрегација на податоците

За да се овозможи ефикасен пристап и пребарување по различни критериуми, имплементирани се неколку модели на агрегација на податоците. Овие агрегации се дизајнирани за да ги задоволат дефинираните сценарија за прашалници и да ја демонстрираат флексибилноста на Key-Value моделите во справувањето со комплексни податочни барања.

Агрегација во Redis

Агрегациите во Redis се изведени со користење на неговите вградени структури на податоци, кои се оптимизирани за брзи операции со множества и рангирања.

- **Агрегација по Жанр:** За секој уникатен жанр се креира Redis `Set` (збирка), чиј клуч е `genre:ИмеНаЖанр`. Секој член на овој сет е клучот на филмот (`movie:ID`) што припаѓа на даденият жанр. Ова овозможува брзо пронаоѓање на сите филмови во одреден жанр.

- **Агрегација по Актер:** Слично како кај жанровите, за секој главен актер (`Star1`) се креира Redis `Set` со клуч `actor:ИмеНаАктер`, содржејќи ги клучевите на сите филмови во кои актерот глуми. Ова овозможува брз пристап до филмографијата на даден актер.
- **Агрегација по Година на Издавање:** За секоја година на издавање, креиран е Redis `Set` со клуч `year:Година`, кој ги содржи клучевите на сите филмови издадени во таа година.
- **Агрегација за Топ Оценети Филмови (по Жанр):** За рангирање на филмовите по оцена во рамките на жанрот, се користи Redis `Sorted Set` (сортирана збирка), каде клучот е `top_rated:ИмеНаЖанр`, а членовите се `movie:ID` со просечната оцена како резултат.

Агрегација во Riak

Бидејќи Riak KV е Key-Value база на податоци без вградени структури како Redis `Set` или `Sorted Set`, агрегацијата се имплементира со поинаков пристап, искористувајќи ги Riak buckets (корпи) за логичка поделба и рачно управување со колекции од клучеви. Ова претставува **второ ниво/модел на агрегација**, различен од директното користење на специјализирани структури како во Redis.

- **Агрегација по Жанр:** За секој уникатен жанр, се користи посебен bucket со име `genre`. Во овој bucket, клучот е името на жанрот (пр., `Adventure`), а вредноста е **листа од `movie:ID`** кои припаѓаат на тој жанр. Секогаш кога се додава нов филм со даден жанр, се вчитува постоечката листа за тој жанр, се додава новиот `movie:ID` (ако веќе не постои) и потоа целата ажурирана листа се запишува назад во Riak.
- **Агрегација по Актер:** Слично на жанровите, за секој главен актер се користи bucket со име `actor`. Клучот во овој bucket е името на актерот (пр., `Matthew McConaughey`), а вредноста е **листа од `movie:ID`** на филмовите во кои глуми. Логиката на ажурирање е иста како кај жанровите – вчитување, додавање, запишување.
- **Агрегација по Година на Издавање:** За секоја година на издавање, се користи bucket со име `year`. Клучот е годината (пр., `2015`), а вредноста е **листа од `movie:ID`** на филмовите издадени таа година.
- **Агрегација за Топ Оценети Филмови (по Жанр):** За рангирање на филмовите, се користи bucket со име `top_rated`. Клучот во овој bucket е жанрот (пр., `Drama`), а вредноста е **листа од tuples**, каде секој tuple содржи `(movie:ID, vote_average)`. При додавање на нов филм, листата се вчитува, се додава новиот tuple, потоа листата **рачно се сортира** по `vote_average` во опаѓачки редослед и се задржува само топ N број на елементи (пр., топ 100). Целата сортирана и скратена листа потоа се запишува назад. Овој пристап симулира `Sorted Set` функционалност, но бара повеќе пресметувања на страната на апликацијата при секое ажурирање.

Дефинирање на сценарија за пристап до податоци (Прашалници)

За тестирање на перформансите и функционалностите на избраните NoSQL бази на податоци, дефинирано е множество од 9 прашалници. Овие прашалници се поделени во три категории, согласно барањата на задачата и се извршени во двете бази на податоци (Redis и Riak) за компаративна анализа.

Едноставни прашалници (Филтрирања)

Овие прашалници вклучуваат директно пребарување или филтрирање на податоци врз основа на еден критериум.

- Пронаоѓање на сите филмови од одреден жанр (пр. „Action“).
- Пронаоѓање на сите филмови во кои глуми одреден актер (пр. „Tom Hanks“).
- Пронаоѓање на сите филмови издадени во одредена година (пр. „2015“).

Сложени прашалници (Спојување на податоци од повеќе инстанци)

Овие прашалници вклучуваат комбинација на критериуми, користејќи операции на пресек (intersection) помеѓу различни агрегирани сетови.

- Пронаоѓање на филмови во кои глуми одреден актер *и* припаѓаат на одреден жанр (пр. „Tom Hanks“ и „Drama“).
- Пронаоѓање на филмови од одреден жанр *и* издадени во одредена година (пр. „Action“ и „2015“).

Доста сложени прашалници (Агрегирани извештаи)

Овие прашалници вклучуваат употреба на сортирани збирки или агрегациски функции за добивање на рангирани или сумарни податоци.

- Пронаоѓање на топ N најдобро оценети филмови во одреден жанр (пр. топ 5 во „Drama“).
- Пронаоѓање на топ N најдобро оценети филмови во одреден жанр *и* издадени во одредена година (пр. топ 3 во „Romance“ и „2019“).
- Броење на филмови со одреден актер (пр. „Leonardo DiCaprio“).
- Броење на високо оценети акциони филмови (со оцена >= 8.0).

Добиени резултати

Во овој дел детално се објаснува податочното множество што се анализира, се презентираат добиените резултати од спроведените експерименти со Redis и Riak, и се дава компаративна анализа на перформансите на двете бази на податоци.

Резултати од спроведените експерименти

Долунаведените табели ги прикажуваат времињата на извршување (во секунди) за секој од дефинираните прашалници, спроведени во Redis и Riak.

Табела 1: Време на извршување на едноставни прашалници

Прашалник	Пронајдени филмови	Redis Време (s)	Riak Време (s)
[Genre: Action]	1370	1.1277	4.5207
[Actor: Tom Hanks]	39	0.0378	0.1460
[Year: 2015]	198	0.1609	0.6377

Табела 2: Време на извршување на сложени прашалници

Прашалник	Пронајдени филмови	Redis Време (s)	Riak Време (s)
[Actor: Tom Hanks AND Genre: Drama]	20	0.0170	0.0712
[Genre: Action AND Year: 2015]	51	0.0468	0.1701

Табела 3: Време на извршување на агрегирани прашалници

Прашалник	Пронајдени филмови	Redis Време (s)	Riak Време (s)
Top 5 rated movies in Genre 'Drama'	5	0.0050	0.0188
Top 3 rated movies in Genre 'Romance' (2019)	3	0.0085	0.0184
Number of movies with actor 'Leonardo DiCaprio'	20	0.0010	0.0028
Number of Action movies with rating >= 8.0	40	0.0010	0.0033

Дискусија за добиените резултати

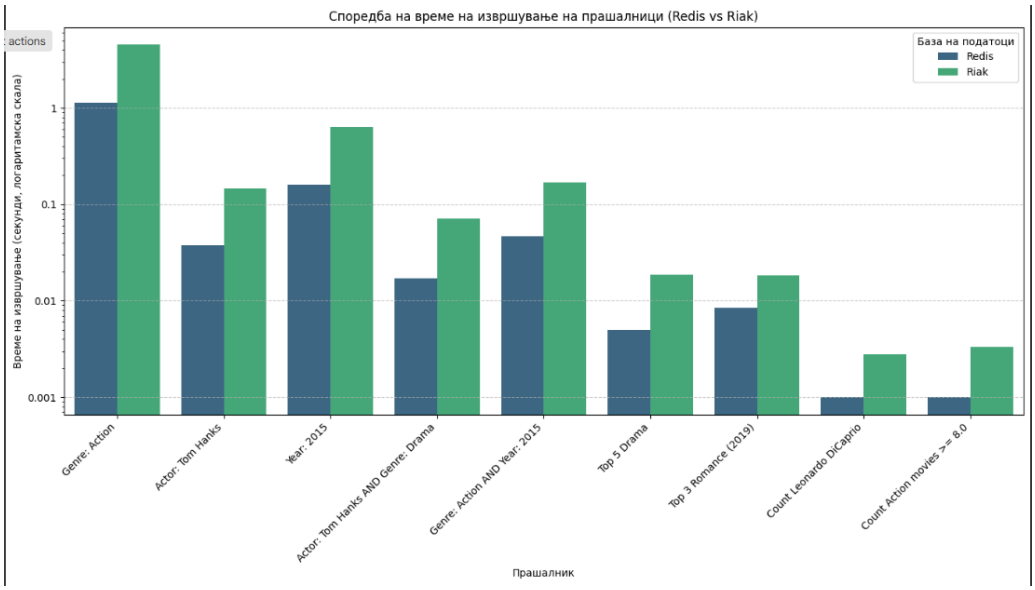
При анализата на добиените резултати, јасно се забележуваат разлики во перформансите помеѓу Redis и Riak за сите категории на прашалници. Генерално, **Redis покажува значително подобри перформанси (помали времиња на извршување) во споредба со Riak** за истите прашалници и истото податочно множество.

- **Едноставни прашалници (Филтрирања):** Кај едноставните прашалници, Redis е конзистентно побрз. Ова е очекувано, бидејќи Redis е in-memory база на податоци оптимизирана за исклучително брз пристап. Неговите вградени структури како `Set` овозможуваат O(1) или O(N) операции (каде N е бројот на елементи во сетот, а не вкупниот број на податоци), што го прави многу ефикасен за операции како `SMEMBERS`. Riak, од друга страна, иако е исто така Key-Value база, неговиот пристап до податоците и симулацијата на сетови преку вчитување и манипулирање со листи од страна на апликацијата (како што е имплементирано), е помалку оптимизиран за такви брзи lookup-и. Конкретно, кај "[Genre: Action]", Redis е приближно 4 пати побрз од Riak.
- **Сложени прашалници (Спојување/Пресеци):** Разликата во перформансите се забележува и кај сложените прашалници кои вклучуваат пресек на податоци од повеќе агрегации (`SINTER` во Redis). Redis `SINTER` е високо оптимизирана команда која извршува операција на пресек директно на серверот. Во Riak, за да се постигне сличен ефект, потребно е да се вчитаат двете листи од `movies` ID за дадените критериуми, потоа да се изврши пресек на тие листи на страната на апликацијата. Ова ја зголемува мрежната комуникација (за вчитување на листите) и процесорската моќ на апликацијата за операцијата на пресек, што резултира со значително подолго време на извршување за Riak (пр., 4-5 пати побавно кај "[Actor: Tom Hanks AND Genre: Drama]").

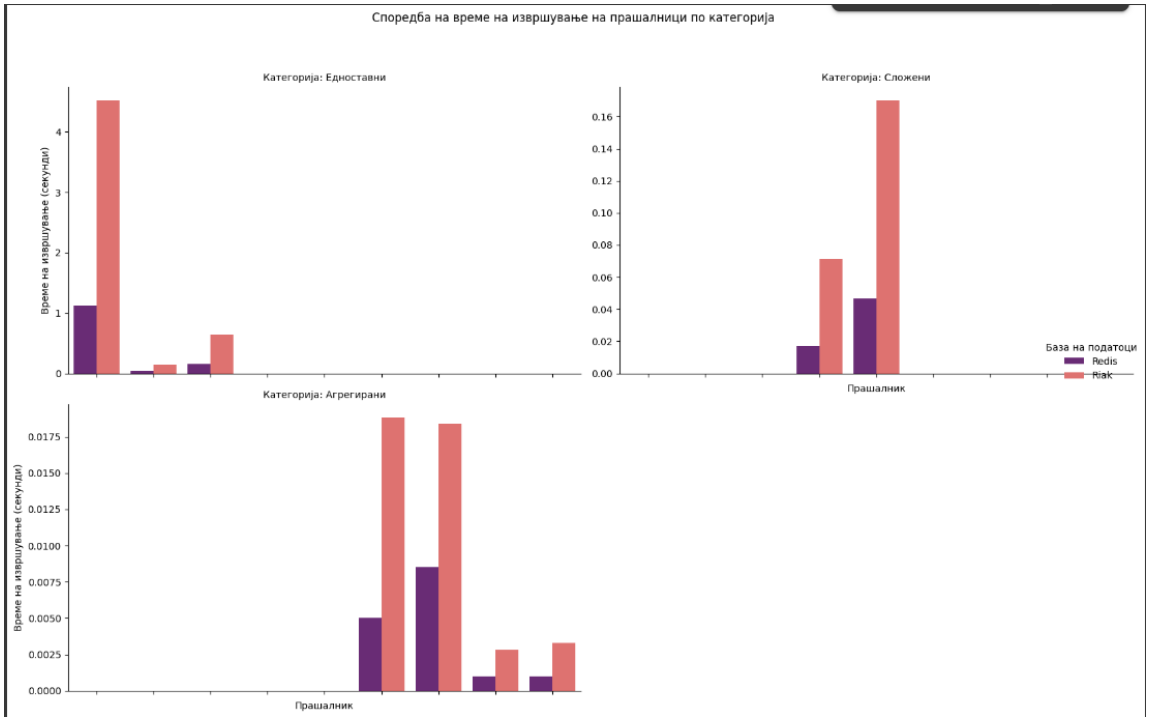
- **Агрегирани прашалници:** Кај агрегираните извештаи, Redis повторно демонстрира супериорни перформанси, особено поради неговите `Sorted Set` структури. `ZREVRANGE` во Redis овозможува многу брзо враќање на рангирани елементи, бидејќи податоците се веќе сортирани и индексирани од самата база. Во Riak, симулацијата на Sorted Set бара рачно сортирање на листата од tuples на страната на апликацијата при секое ажурирање и при секое пребарување на врвни елементи. Разликите се помали отколку кај сложените прашалници, но Redis е сепак 2-4 пати побрз. За операциите како броење членови на сет (`SCARD` во Redis), Redis е многу брз бидејќи тоа е директна операција на неговата структура. Во Riak, ова вклучува вчитување на целата листа и броење на нејзините елементи, што е малку побавно.

Визуелизација

Слика 1: Споредба на време на извршување на прашалници (Redis vs Riak)



Слика 2: Споредба на време на извршување на прашалници по категорија



Заклучок

Преку оваа проектна задача детално ги истраживме Key-Value NoSQL базите на податоци, фокусирајќи се на Redis и Riak. Главната цел беше да се анализира, моделира и имплементира складирање и агрегација на филмски метаподатоци, а потоа да се изврши компаративна анализа на перформансите преку дефинирано множество прашалници.

Успешно имплементиравме различни модели на агрегација во двете бази: во Redis искористивме вградени структури како Set и Sorted Set за ефикасни операции, додека во Riak применивме пристап со рачно управувани листи во посебни buckets за да ги симулираме овие функционалности.

Резултатите од перформансните мерења јасно покажаа дека **Redis е значително побрз** за сите типови прашалници поради неговата in-memory природа и оптимизирани структури. Riak, иако робустен за дистрибуирани системи, покажа поголемо временско задоцнување, што делумно се должи на дополнителното оптоварување од агрегацијата на апликациско ниво. Ова ја потврдува важноста од избор на база на податоци според специфичните барања за брзина и скалабилност.

Како можности за идно подобрување може да се искористи Riak Secondary Indexes (2i) и MapReduce функционалности за поефикасни агрегации, тестирање на поголеми податочни множества, како и проширување на компарацијата со други Key-Value бази во дистрибуирани околини.

Користена литература

https://docs.riak.com/riak/kv/2.2.3/index.html

https://redis.io/docs/latest/develop/

https://www.kaggle.com/datasets/shubhamchandra235/imdb-and-tmdb-movie-metadata-big-dataset-1m