به نام خدا

جستجو بر اساس متن، مانند جستجوی تمام‌متن در Elasticsearch، تمرکز زیادی بر روی تطابق دقیق "متن به متن" دارد. این نوع جستجو به خوبی برای پایگاه‌های داده‌ای که اطلاعات را به صورت متنی ذخیره می‌کنند، کاربردی است و می‌تواند نتایج دقیقی را بر اساس تطابق‌های متنی ارائه دهد.

از سوی دیگر، وکتور دیتابیس‌ها (vector database) مانند Weaviate، با استفاده از تکنولوژی‌های پیشرفته‌ای مانند بردارهای معنایی، قادر به انجام جستجوهای معنایی هستند. این نوع جستجو به جای تمرکز بر تطابق‌های دقیق متنی، به تحلیل و درک معانی و مفاهیم داده‌ها پرداخته و به این ترتیب قادر است نتایج مرتبط‌تری را در موقعیت‌های پیچیده‌تر ارائه دهد.

در جستجو و ایندکس‌گذاری داده‌ها، علاوه بر امکان جستجو بر اساس نام مکان (place_name)، می‌توان از موقعیت جغرافیایی نیز بهره برد. این امکان به ویژه زمانی مفید است که نیاز به یافتن مکان‌ها یا داده‌ها بر اساس فاصله مکانی وجود دارد. برای این منظور، از الگوریتم geo-distance استفاده می‌شود. geo-distance معمولاً با استفاده از فرمول‌های ریاضی مانند "Haversine formula" یا "Vincenty formula" برای محاسبه فاصله بر روی سطح کره زمین انجام می‌شود. این فرمول‌ها به درستی فاصله‌های واقعی بین نقاط جغرافیایی را با توجه به انحنای زمین محاسبه می‌کنند.

Geo-distance به ما این امکان را می‌دهد که نتایج جستجو را بر اساس فاصله جغرافیایی فیلتر کنیم. به عبارت دیگر، می‌توانیم جستجوی خود را به مکان‌هایی که در شعاع مشخصی از یک نقطه جغرافیایی خاص قرار دارند، محدود کنیم. این ویژگی به ویژه در برنامه‌های کاربردی که نیاز به شناسایی نزدیک‌ترین مکان‌ها یا خدمات در نزدیکی یک موقعیت جغرافیایی دارند، بسیار مفید است.

هر دو دیتابیس Elasticsearch و Weaviate از این قابلیت پشتیبانی می‌کنند:

- **Elasticsearch**: با استفاده از ویژگی geo_distance، می‌توان جستجوهایی را بر اساس فاصله مکانی از موقعیت مشخص انجام داد و نتایج را با دقت فیلتر کرد.
- **Weaviate**: نیز از این قابلیت بهره‌مند است و می‌تواند با استفاده از WithinGeoRange جستجوهای معنایی و جغرافیایی را به صورت ترکیبی انجام دهد.

به این ترتیب، با استفاده از قابلیت‌های geo-distance، می‌توان جستجوهای پیچیده‌تری را انجام داد که هم به تطابق‌های متنی و هم به موقعیت‌های جغرافیایی توجه دارند، و از هر دو پایگاه داده به طور مؤثر بهره برد.

در این گزارش، به بررسی و تحلیل دو میکروسرویس پرداخته شده است که برای قابلیت‌های جستجو و ایندکس‌گذاری داده‌ها استفاده می‌شوند: Weaviate و Elasticsearch. هر یک از این میکروسرویس‌ها از تکنولوژی‌های خاصی برای ارائه ویژگی‌های جستجو و ایندکس‌گذاری بهره می‌برد.

۱. Weaviate

Weaviate یک پایگاه داده برداری (vector database) است که به‌ویژه برای جستجوی معنایی (semantic search) و مدیریت داده‌های برداری طراحی شده است. این میکروسرویس با استفاده از تبدیل متنی به بردارها (text2vec-transformers) و الگوریتم‌های پیشرفته، قابلیت‌های جستجوی معنایی را ارائه می‌دهد. ویژگی‌های اصلی Weaviate به شرح زیر است:

- **الگوریتم HNSW:** در Weaviate برای جستجوی نزدیک‌ترین همسایه‌ها از الگوریتم Hierarchical Navigable Small World استفاده می‌کند. این الگوریتم به‌طور مؤثر برای جستجوی برداری در فضاهای با ابعاد بالا طراحی شده و امکان جستجو در مقیاس بزرگ را فراهم می‌آورد.
- **بردار جستجو (Vector Search):** دیتابیس Weaviate با استفاده از بردارهای متنی، جستجوی معنایی را انجام می‌دهد که به کاربران این امکان را می‌دهد تا اطلاعات مرتبط را بر اساس مفهوم و معنی جستجو کنند.
- **جستجوی جغرافیایی (Geo-Search):** با استفاده از فیلد geoCoordinates, Weaviate امکان جستجوی مبتنی بر مکان را فراهم می‌آورد، که برای کاربردهایی که نیاز به جستجو در موقعیت‌های جغرافیایی دارند، مناسب است.

الگوریتم HNSW

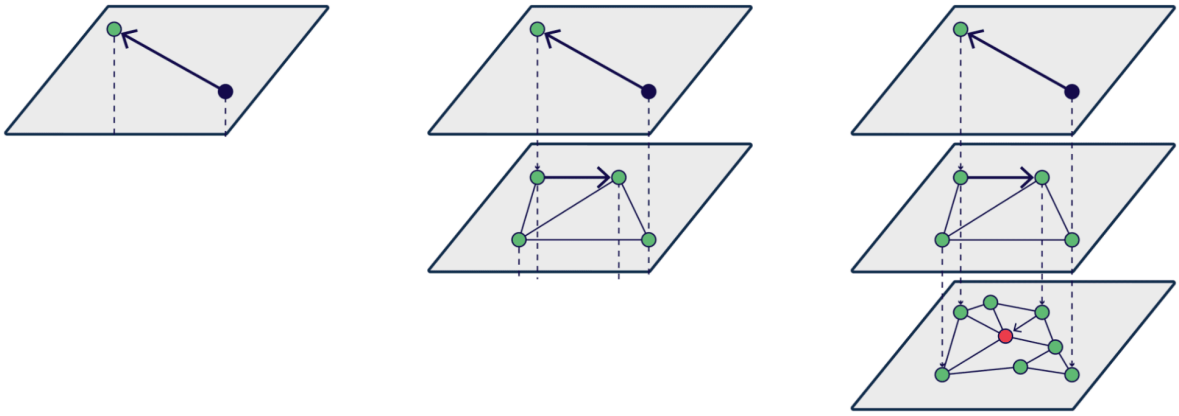
الگوریتم HNSW، که مخفف Hierarchical Navigable Small World است، یکی از الگوریتم‌های پیشرفته برای جستجوی نزدیک‌ترین همسایه‌ها در فضاهای برداری با ابعاد بالا است. این الگوریتم به‌ویژه برای جستجوی سریع و مقیاس‌پذیر در داده‌های برداری طراحی شده است و شامل ویژگی‌های زیر می‌باشد:

- **ساختار چند لایه:** در HNSW، گراف شامل چندین لایه از بردارها است. هر لایه شامل زیرمجموعه‌ای از بردارهای لایه پایین‌تر است. این ساختار به جستجوگر این امکان را می‌دهد که به سرعت منطقه کلی گراف را شناسایی کرده و سپس جستجوی دقیق‌تری را در لایه‌های پایین‌تر انجام دهد. در نهایت، جستجو به لایه پایین‌تر که شامل تمام بردارهای موجود است، منتقل می‌شود.
- **مشابه یک لیست پرش (Skip List):** به‌طور انتزاعی، گراف HNSW مشابه یک لیست پرش با ابعاد بالا است که در آن لایه‌های بالاتر برای جستجوی جهانی و لایه‌های پایین‌تر برای جستجوی محلی استفاده می‌شوند.
- **مقیاس‌پذیری:** الگوریتم HNSW برای مقیاس‌های بزرگ و داده‌های با ابعاد بالا بهینه شده است و می‌تواند به‌طور مؤثری با داده‌های بزرگ و پیچیده تعامل داشته باشد.
- **پیچیدگی زمانی:** پیچیدگی زمانی الگوریتم HNSW به طور تقریبی لگاریتمی است. به بیان دقیق‌تر، زمانی که تعداد نمونه‌ها (نمونه‌های داده) افزایش می‌یابد، تعداد مقایسه‌ها و عملیات مورد نیاز برای جستجو و درج، به صورت لگاریتمی به پایه ۱۰ افزایش می‌یابد. به عبارت دیگر، زمان جستجو به ازای هر نمونه جدید به میزان $\log_{10}(n)$ افزایش می‌یابد، که این امر باعث می‌شود HNSW برای مقیاس‌های بزرگ داده بسیار کارآمد باشد.
- **عملکرد سریع:** به دلیل ساختار سلسله‌مراتبی و گراف‌های قابل پیمایش که در الگوریتم HNSW استفاده می‌شود، جستجو داده‌ها با سرعت بسیار بالا انجام می‌شود. این ویژگی باعث می‌شود که HNSW برای پردازش‌های با حجم بالا و نیاز به جستجوهای سریع بسیار مناسب باشد.

به عنوان مثال:

اگر تعداد نمونه‌ها در پایگاه داده به یک میلیون برسد، پیچیدگی جستجو و درج به طور تقریبی برابر با $\log_{10}(1000000) \approx ۶$ خواهد بود، که نشان‌دهنده کارایی بسیار بالای الگوریتم در مقیاس‌های بزرگ است. قابل ذکر است که زمان تقریبی برای هر سرچ برای ۱ میلیون داده تقریبا ۱ ثانیه است.

در زیر، نمایی از ساختار الگوریتم HNSW و چگونگی عملکرد آن آورده شده است:



HNSW

پیاده‌سازی ایندکس در Weaviate:

در Weaviate، با تعریف یک کلاس به نام Document, فیلدهای مختلفی مانند نام مکان، موقعیت جغرافیایی، آدرس و برچسب‌ها تعریف شده‌اند. این فیلدها به سیستم این امکان را می‌دهند که داده‌ها را بر اساس معیارهای مختلفی جستجو و ایندکس کند.

جستجو در Weaviate:

تابع جستجو در Weaviate به گونه‌ای طراحی شده است که کاربران می‌توانند با تعیین نام مکان و مختصات جغرافیایی، نتایج را بر اساس نزدیک‌ترین موقعیت‌های جغرافیایی جستجو کنند. این جستجو به طور مؤثری امکان فیلتر کردن نتایج بر اساس فاصله و تطابق نام مکان را فراهم می‌کند.

۲. Elasticsearch

Elasticsearch یک موتور جستجوی قدرتمند است که بر پایه کتابخانه Lucene ساخته شده و برای جستجوهای متنی و تجزیه و تحلیل داده‌ها طراحی شده است. این میکروسرویس با استفاده از ایندکس‌های معکوس (Inverted Indexes) و قابلیت‌های جستجوی پیشرفته، به کاربران این امکان را می‌دهد که داده‌ها را با سرعت و دقت بالا جستجو کنند.

- **جستجوی متنی کامل (Full-Text Search):** در Elasticsearch برای جستجوهای متنی و تجزیه و تحلیل داده‌ها با استفاده از ساختار ایندکس معکوس، عملکرد بسیار بالایی دارد.
- **مقیاس‌پذیری (Scalability):** در Elasticsearch به‌خوبی مقیاس‌پذیر است و قادر است با حجم‌های بزرگ داده‌ها به‌طور مؤثر عمل کند.
- **جستجوی جغرافیایی (Geo-Search):** دیتابیس Elasticsearch همچنین از فیلد geo_point برای جستجوی داده‌های جغرافیایی پشتیبانی می‌کند و می‌تواند نتایج را بر اساس موقعیت‌های جغرافیایی فیلتر کند.

پیاده‌سازی ایندکس در Elasticsearch:

در Elasticsearch، ایندکس places با ویژگی‌های مختلفی تعریف شده است که شامل نام مکان، موقعیت جغرافیایی، آدرس و برچسب‌ها می‌باشد. این ساختار به Elasticsearch این امکان را می‌دهد که داده‌ها را بر اساس نیازهای جستجو و ایندکس‌گذاری متنوعی مدیریت کند.

جستجو در Elasticsearch:

تابع جستجو در Elasticsearch به کاربران این امکان را می‌دهد که با استفاده از نام مکان، برچسب و موقعیت جغرافیایی، نتایج جستجو را فیلتر کنند. این جستجو با استفاده از ویژگی‌هایی مانند multi-match و geo_distance، به‌طور مؤثری داده‌ها را بر اساس نیازهای جستجو فیلتر می‌کند.

نحوه ایندکس‌سازی و جستجو در Elasticsearch

۱. ساخت ایندکس در Elasticsearch

برای ایجاد یک ایندکس جدید در Elasticsearch، نیاز به تعریف ساختار داده‌ها (مپینگ‌ها) داریم. در اینجا، یک ایندکس با نام places تعریف شده است که شامل چهار فیلد است:

1. **place_name:** فیلدی از نوع text که برای جستجو در نام مکان‌ها استفاده می‌شود. نوع text به این معنی است که داده‌ها به صورت متنی تجزیه و تحلیل می‌شوند و مناسب برای جستجوی متنی است.
2. **location:** فیلدی از نوع geo_point برای ذخیره مختصات جغرافیایی. این نوع داده برای جستجوی مبتنی بر مکان استفاده می‌شود.
3. **address:** فیلدی از نوع text که برای ذخیره آدرس‌ها استفاده می‌شود. مشابه فیلد place_name, این فیلد نیز به صورت متنی تجزیه و تحلیل می‌شود.
4. **tag:** فیلدی از نوع keyword برای ذخیره برچسب‌ها. نوع keyword برای داده‌های دقیق و غیرقابل تجزیه و تحلیل مناسب است.

تعریف ایندکس به صورت زیر است:

```
index_name = 'places'

body = {

  "mappings": {

    "properties": {

      "place_name": {

        "type": "text",

      },
```

```
        "location": {
            "type": "geo_point"
        },
        "address": {
            "type": "text"
        },
        "tag": {
            "type": "keyword"
        }
    }
  }
}
```

۲. جستجو در Elasticsearch

برای جستجو در ایندکس، از تابع multi_match_search استفاده می‌شود. این تابع به دنبال تطابق‌های متنی و جغرافیایی در ایندکس است. تابع به شرح زیر عمل می‌کند:

1. **پارامترهای جستجو:**
   - **place_name**: برای جستجوی نام مکان‌ها.
   - **tag**: برای جستجوی برچسب‌ها.
   - **location**: برای فیلتر کردن نتایج بر اساس فاصله جغرافیایی.
2. **ساختار جستجو:**
   - **must**: جستجوهایی که باید حتماً تطابق داشته باشند. در اینجا، جستجوی بر اساس نام مکان (place_name) است.
   - **should**: جستجوهایی که مطلوب است تطابق داشته باشند. در اینجا، جستجوی بر اساس برچسب (tag) است.
   - **filter**: فیلتر کردن نتایج بر اساس شرایط مشخص. در اینجا، نتایج بر اساس فاصله جغرافیایی از موقعیت مشخص شده (location) فیلتر می‌شود. (۲۰۰ متر)
3. **کد جستجو:**

```
search_query = {
```

```
  "query": {
    "bool": {
      "must": [
        {"match": {"place_name": data.place_name}}
      ],
      "should": [
        {"match": {"tag": data.tag}}
      ],
      "filter": [
        {
          "geo_distance": {
            "distance": "0.2km",
            "location": data.location
          }
        }
      ]
    }
  }
}
results = self.client.search(index=index_name, body=search_query, size='1000')
```

تست locust

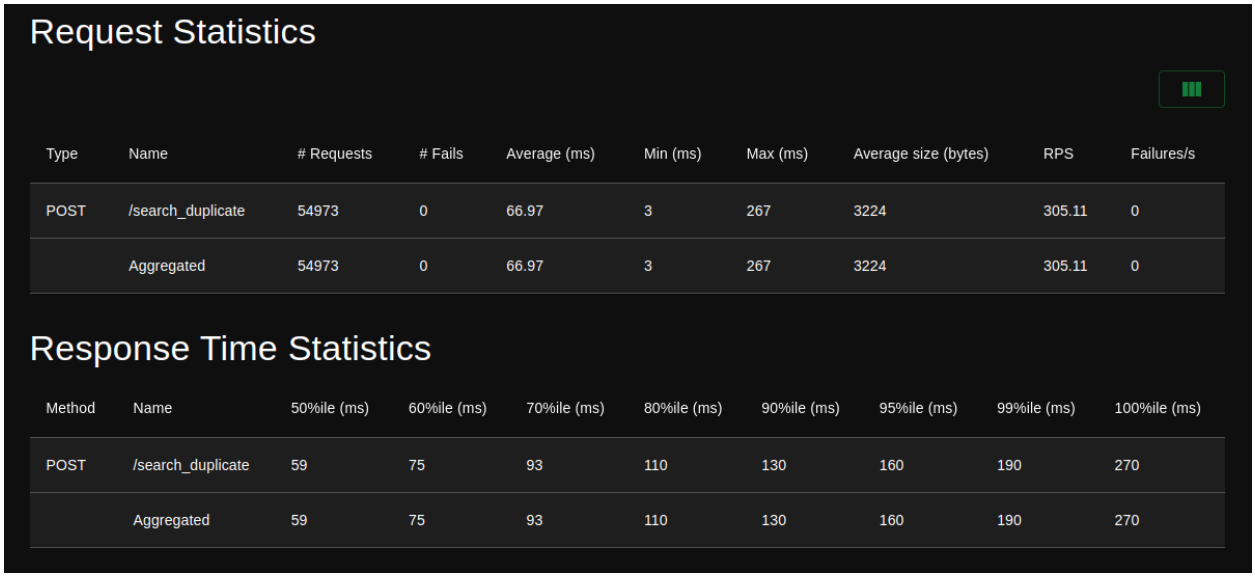
تست لوکاست یک ابزار قوی برای ارزیابی عملکرد و مقیاس‌پذیری سیستم‌ها است. این ابزار به ما این امکان را می‌دهد که بارهای مختلف را شبیه‌سازی کرده و عملکرد سیستم را تحت فشارهای متفاوت مورد بررسی قرار دهیم. در این تست خاص، از لوکاست برای ارزیابی عملکرد سیستم جستجوی تکراری استفاده شد. سناریوی تست به گونه‌ای طراحی شده است که بار کاربران را به تدریج افزایش دهد تا بتوان عملکرد سیستم را در شرایط مختلف سنجید.

۱. سناریو:

در این تست، دو مرحله بارگذاری با دوره‌های زمانی مشخص تعریف شده است. در مرحله اول، سیستم با ۱۰۰ کاربر و نرخ ایجاد ۱۰ کاربر در ثانیه تحت فشار قرار گرفته است. در مرحله دوم، تعداد کاربران به ۱۰۰۰ افزایش یافته و نرخ ایجاد کاربران همچنان ۱۰ کاربر در ثانیه حفظ شده است. هدف از این سناریو، ارزیابی نحوه عملکرد سیستم در برابر بارهای مختلف و اندازه‌گیری پاسخ‌دهی آن تحت شرایط فشار بالا بوده است.

۲. نتایج:

نتایج تست نشان می‌دهد که مجموعاً ۵۴۹۷۳ درخواست به endpoint /search_duplicate ارسال شده که هیچ‌یک از آن‌ها با خطا مواجه نشده است. زمان پاسخگویی میانگین ۶۶/۹۷ میلی‌ثانیه بوده و بیشترین زمان پاسخگویی ۲۶۷ میلی‌ثانیه ثبت شده است. زمان پاسخگویی ۹۵ درصد از درخواست‌ها زیر ۱۶۰ میلی‌ثانیه بوده و نرخ پردازش حدود ۳۰۵.۱۱ درخواست در ثانیه بدون شکست در درخواست‌ها، نشان‌دهنده عملکرد پایدار و کارآمد سیستم تحت بار بالا است.



Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/search_duplicate	54973	0	66.97	3	267	3224	305.11	0
	Aggregated	54973	0	66.97	3	267	3224	305.11	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/search_duplicate	59	75	93	110	130	160	190	270
	Aggregated	59	75	93	110	130	160	190	270

نحوه ایندکس‌سازی و جستجو در Weaviate

۱. ساخت ایندکس در Weaviate

برای ایجاد یک کلاس جدید در Weaviate، باید ویژگی‌ها و نوع داده‌های آن را تعریف کنیم.

برای درج داده‌ها در Weaviate، به ویژه هنگامی که از ویژگی موقعیت جغرافیایی استفاده می‌کنیم، باید اطلاعات موقعیت را به شکل خاصی فرمت کنیم. به طور خاص، برای فیلد location که از نوع geoCoordinates است، باید مختصات جغرافیایی به صورت زیر ارائه شود:

```
"location": {

   "latitude": 35.7774394,

   "longitude": 51.355904

}
```

بنابراین، برای آماده‌سازی دیتاست و انجام عملیات ایندکسینگ، داده‌ها را به این فرمت تبدیل کرده و سپس ایندکسینگ را مطابق با این ساختار انجام داده شده. این کار اطمینان حاصل می‌کند که موقعیت‌های جغرافیایی به درستی در پایگاه داده ذخیره و مورد استفاده قرار می‌گیرند.

در اینجا، یک کلاس با نام Document تعریف شده است که برای ذخیره مکان‌ها با ویژگی‌های مختلف استفاده می‌شود:

1. **place_name:** فیلدی از نوع text که نام مکان را ذخیره می‌کند. این نوع داده برای جستجوهای متنی مناسب است.
2. **location:** فیلدی از نوع geoCoordinates که مختصات جغرافیایی (عرض جغرافیایی و طول جغرافیایی) را ذخیره می‌کند. این نوع داده برای جستجوهای جغرافیایی استفاده می‌شود.
3. **address:** فیلدی از نوع text که آدرس مکان را ذخیره می‌کند. مشابه فیلد place_name, این فیلد نیز برای جستجوی متنی مناسب است.
4. **tag:** فیلدی از نوع text که برچسب یا دسته‌بندی مربوط به مکان را ذخیره می‌کند. این فیلد برای جستجوی متنی استفاده می‌شود.

تعریف کلاس به صورت زیر است:

```
document_class = {
  "class": "Document",
  "description": "A class representing locations with names, addresses, and tags",
  "vectorizer": "text2vec-transformers",
  "properties": [
    {
      "name": "place_name",
```

```
        "dataType": ["text"],
        "description": "The name of the place"
      },
      {
        "name": "location",
        "dataType": ["geoCoordinates"],
        "description": "The geographical location (latitude and longitude)"
      },
      {
        "name": "address",
        "dataType": ["text"],
        "description": "The address of the place"
      },
      {
        "name": "tag",
        "dataType": ["text"],
        "description": "The category or tag associated with the place"
      }
    ]
}
```

۲. جستجو در Weaviate

برای جستجو در Weaviate، از تابع search_near_geo_raw استفاده می‌شود. این تابع جستجوی مکانی نزدیک به مختصات جغرافیایی را انجام می‌دهد و نتایج را بر اساس تطابق نام مکان و فاصله جغرافیایی فیلتر می‌کند. تابع به شرح زیر عمل می‌کند:

1. **پارامترهای جستجو:**
   - **class_name**: نام کلاسی که جستجو در آن انجام می‌شود.
   - **data**: داده‌های ورودی شامل نام مکان و مختصات جغرافیایی.
   - **max_distance**: حداکثر فاصله برای جستجو به کیلومتر (۲۰۰ متر).
2. **ساختار جستجو:**
   - **where**: شرایط جستجو را تعریف می‌کند. این شرایط شامل تطابق نام مکان و فیلتر کردن نتایج بر اساس فاصله جغرافیایی است.
   - **operator: Equal**: جستجو بر اساس نام مکان.
   - **operator: WithinGeoRange**: فیلتر کردن نتایج بر اساس فاصله جغرافیایی از موقعیت داده شده.
3. **کد جستجو:**

```
query = f"""
```

```
    {{
      Get {{
        {class_name}(
        where: {{
          operator: And,
          operands: [
          {{
            operator: Equal,
            path: ["place_name"],
            valueText: "{data.place_name}"
          }},
          {{
            operator: WithinGeoRange,
            valueGeoRange: {{
              geoCoordinates: {{
                latitude: {data.location.latitude},
                longitude: {data.location.longitude}
              }},
              distance: {{
                max: {max_distance}
              }}
            }},
            path: ["location"]
          }}
          ]
        }}
        ) {{
        place_name
        address
        location {{
          latitude
          longitude
        }}
        tag
        }}
      }}
    }}
    """
results = self.client.query.raw(query)
```

تست locust

۱. سناریو:

در این تست، دو مرحله بارگذاری با دوره‌های زمانی مشخص تعریف شده است. در مرحله اول، سیستم با ۱۰۰ کاربر و نرخ ایجاد ۱۰ کاربر در ثانیه تحت فشار قرار گرفته است. در مرحله دوم، تعداد کاربران به ۱۰۰۰ افزایش یافته و نرخ ایجاد کاربران همچنان ۱۰ کاربر در ثانیه حفظ شده است. هدف از این سناریو، ارزیابی نحوه عملکرد سیستم در برابر بارهای مختلف و اندازه‌گیری پاسخ‌دهی آن تحت شرایط فشار بالا بوده است.

۲. نتایج:

مجموعاً ۵۶۱۸۸ درخواست به endpoint /search_duplicate ارسال شده که هیچ‌یک از آن‌ها با خطا مواجه نشده است. زمان پاسخگویی میانگین ۳۹/۵۵ میلی‌ثانیه بوده و بیشترین زمان پاسخگویی ۲۰۹ میلی‌ثانیه ثبت شده است. زمان پاسخگویی ۹۵ درصد از درخواست‌ها زیر ۱۱۰ میلی‌ثانیه و ۹۹ درصد زیر ۱۴۰ میلی‌ثانیه بوده است. نرخ پردازش حدود ۳۱۱.۷۴ درخواست در ثانیه بدون شکست در درخواست‌ها، نشان‌دهنده عملکرد بسیار سریع و کارآمد سیستم است که به خوبی قادر به مدیریت بار بالا و ارائه پاسخ‌های سریع در شرایط فشار می‌باشد.

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/search_duplicate	56188	0	39.55	3	209	3374	311.74	0
	Aggregated	56188	0	39.55	3	209	3374	311.74	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/search_duplicate	30	40	51	64	85	110	140	210
	Aggregated	30	40	51	64	85	110	140	210

نتیجه گیری

در مقایسه نتایج تست لوکاست برای Elasticsearch و Weaviate، مشاهده می‌شود که هر دو سیستم عملکرد قابل قبولی دارند، اما با تفاوت‌هایی در زمان پاسخگویی. برای Elasticsearch، میانگین زمان پاسخگویی ۶۶/۹۷ میلی‌ثانیه و بیشترین زمان ۲۶۷ میلی‌ثانیه بوده است، در حالی که Weaviate با میانگین زمان پاسخگویی ۳۹/۵۵ میلی‌ثانیه و حداکثر زمان ۲۰۹ میلی‌ثانیه، سریع‌تر عمل کرده است. همچنین، ۹۵ درصد از درخواست‌های Weaviate زیر ۱۱۰ میلی‌ثانیه پاسخ داده شده‌اند، در مقایسه با زمان زیر ۱۶۰ میلی‌ثانیه برای

Elasticsearch. از نظر نرخ پردازش، Weaviate با ۳۱۱.۷۴ درخواست در ثانیه نسبت به ۳۰۵.۱۱ درخواست در ثانیه برای Elasticsearch، عملکرد مشابه و کمی بهتری را ارائه داده است. این مقایسه نشان می‌دهد که Weaviate توانایی بهتری در پاسخ‌دهی سریع‌تر و مدیریت بار بالاتر نسبت به Elasticsearch دارد.

در بررسی کلی از عملکرد و قابلیت‌های الگوریتم‌های ایندکس‌گذاری، هر دو میکروسرویس Weaviate و Elasticsearch ویژگی‌های منحصر به فردی را ارائه می‌دهند. Weaviate از الگوریتم HNSW برای ایندکس‌گذاری و جستجو استفاده می‌کند که به دلیل ساختار لایه‌ای و اتصالات گرافی، امکان جستجوی سریع و دقیق را در فضاهای با ابعاد بالا فراهم می‌آورد. از طرف دیگر، Elasticsearch با استفاده از ساختار ایندکس معکوس (Inverted Index) برای جستجوی متنی، توانایی بالایی در پردازش سریع و دقیق درخواست‌ها دارد. نتایج تست‌های لوکاست نشان می‌دهد که هر دو سیستم به خوبی می‌توانند بارهای بالا را مدیریت کنند، اما Weaviate با زمان پاسخگویی میانگین کمتر و نرخ پردازش بالاتر، عملکردی سریع‌تر و کارآمدتر از Elasticsearch از خود نشان داده است.