Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Možnosti systému Elasticsearch

2024

Martin Polášek

Vedoucí práce:
Mgr. Jan Laštovička, Ph.D.

Studijní program: Informační technologie,
prezenční forma

Bibliografické údaje

Autor:	Martin Polášek
Název práce:	Možnosti systému Elasticsearch
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Informační technologie, prezenční forma
Vedoucí práce:	Mgr. Jan Laštovička, Ph.D.
Počet stran:	32
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Martin Polášek
Title:	Options of Elasticsearch system
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Information Technologies, full-time form
Supervisor:	Mgr. Jan Laštovička, Ph.D.
Page count:	32
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Má bakalářská práce se zabývá představením a možnostmi distribuovaného vyhledávacího a analytického systému Elasticsearch s full-textovým vyhledáváním. Rozebírá jeho využití, nastavení a hlavní funkcionality. Popisuje strukturu systému, indexaci dat a dotazovací jazyk Query DSL. Na historických meteorologických datech analyzuje přínos použitých typů mezipamětí a jak rychle systém reaguje na některé vyhledávací dotazy zaměřené na geografické a datové filtry s náhodně vybranými parametry v mezích dat. Na závěr popisuje mnou vyvinutou aplikaci, která posloužila k měření doby vyhodnocení dotazů.

Synopsis

My bachelor's thesis deals with the introduction and possibilities of the distributed search and analytical system Elasticsearch with full-text search capabilities. It discusses its utilization, configuration, and main functionalities. It describes the system's structure, data indexing, and the Query DSL query language. Using historical meteorological data, it analyzes the benefits of the utilized caching mechanisms and how quickly the system responds to some search queries focused on geographic and date filters with randomly chosen parameters within the data bounds. Finally, it describes the application I developed, which was used to measure the query evaluation time.

Klíčová slova: Elasticsearch; struktura systému; analýza dotazů; mezipaměť; filtrace; čas vyhledání;

Keywords: Elasticsearch; system structure; query analysis; cache; filtration; search time;

Za všechnu poskytnutou pomoc a věnovaný čas bych chtěl moc poděkovat mému vedoucímu práce panu Mgr. Janu Laštovičkovi, Ph.D. a ještě bych chtěl poděkovat mé rodině, která mi byla po dobu psaní práce oporou.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Systém Elasticsearch

Elasticsearch je distribuovaný [1] vyhledávací a analytický systém, který umožňuje i fulltextové vyhledávání. Je založený na knihovně Lucene. Fulltextové vyhledávání je vyhledávání, které prochází celý obsah textového souboru. Obsah je indexován a po jeho analýze je vytvořena pomocná datová struktura, která se používá k následnému vyhledávání.

Knihovna Lucene je open-source vyhledávací knihovna napsaná v Javě. Byla přenesena i do dalších jazyků, jako jsou Python, C#, C++ a další. Je podporovaná Apache Software Foundation a je vydaná pod Apache Software License.

Elasticsearch server je naprogramovaný v Javě a ke komunikaci s klientem využívá JSON a RESTful API. Elasticsearch klient je dostupný v mnoha programovacích jazycích. Elasticsearch je distribuován zdarma pod open-source licencí Apache a je vyvíjen společně se systémy pro sběr dat a analýzu logů, jmenující se Logstash a Beats, a s analytickou a vizualizační platformou Kibana. Společně tvoří integrovaný systém Elastic Stack. [2]

1.1 Základy Elasticsearch

Elasticsearch může být použit k vyhledávání v jakémkoliv typu textového souboru. Umožňuje škálovatelné vyhledávání, to znamená, že je možné zvyšovat množství požadavků na vyhledávání bez ztráty výkonu a spolehlivosti. Data jsou uložena v *indexech* (*index*), které lze považovat za optimalizovanou kolekci *dokumentů* (*document*), kde každý dokument je kolekcí *položek* (*field*), což jsou páry klíč–hodnota, které obsahují data. Podporuje i distribuovanou architekturu. Ve výchozím nastavení Elasticsearch indexuje všechna data v každé položce a každá indexovaná položka má vyhrazenou optimalizovanou datovou strukturu. Například textové položky jsou uloženy v invertovaných indexech a numerické a geografické položky jsou uloženy ve stromech BKD. Schopnost používat datové struktury podle typů položek k sestavení a vrácení výsledků vyhledávání je to, co dělá Elasticsearch tak rychlým.

Index je ve skutečnosti jen logické seskupení jednoho nebo více *fyzických fragmentů* (*physical shards*) , kde každý *fragment* (*shard*) je vlastně samostatný index. Fragment se dále dělí na *segmenty*. Segment je v Elasticsearch komplexní datová struktura, která zahrnuje informace o indexovaných dokumentech, tokenizovaných termínech, statistikách a dalších metadatech potřebných pro efektivní vyhledávání.

Celý systém může být distribuovaný mezi *uzly* (*nodes*), které spolu spolupracují v jednom *svazku* (*cluster*) (viz obr. 1). Distribucí dokumentů v indexu mezi více fragmentů a distribucí těchto fragmentů mezi více uzlů může Elasticsearch zajistit redundanci. Redundance chrání před selháním hardwaru a zvyšuje kapacitu dotazů při přidávání uzlů do svazku. Jak svazek roste (nebo se zmenšuje),

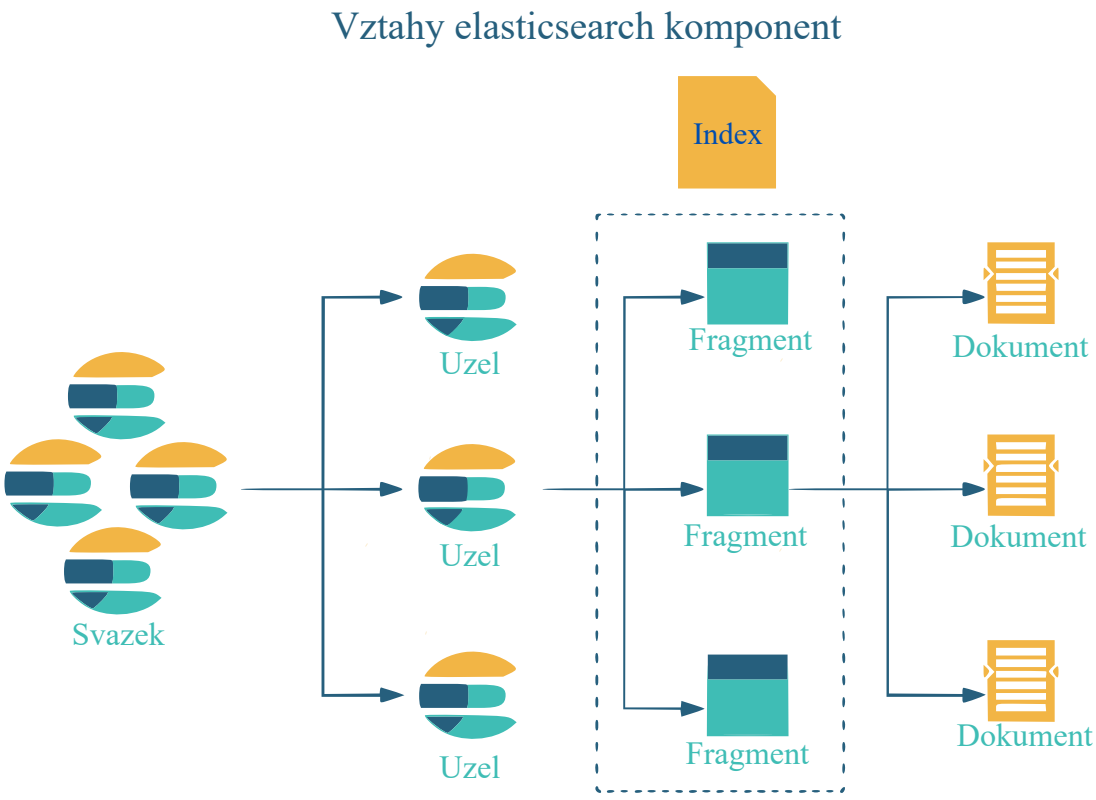
[1] Systém běží na několika spolupracujících serverech.

[2] Informace v práci jsou čerpány z `https://www.elastic.co/`

Elasticsearch automaticky migruje fragmenty, aby svazek znovu vyvážil.

Existují dva typy fragmentů: *primární* (*primary*) a *repliky* (*replica*). Každý dokument v indexu patří do jednoho primárního fragmentu. Replika datového fragmentu je kopií primárního datového fragmentu. Repliky poskytují redundantní kopie dat kvůli ochraně před selháním hardwaru a zvyšují kapacitu pro obsluhu požadavků na čtení, jako je vyhledávání nebo načítání dokumentu.

Počet primárních fragmentů v indexu je pevně stanoven v okamžiku vytvoření indexu, ale počet replikovaných fragmentů lze kdykoli změnit, aniž by došlo k přerušení operací indexování nebo dotazování. Každý elasticsearch uzel má jeden nebo více fragmentů a jedná jako koordinátor, deleguje operace příslušným fragmentům. Data, která mají mezi sebou nějakou souvislost jsou obvykle uložena ve stejném indexu, který se skládá z jednoho nebo více hlavních fragmentů a žádného nebo více replikovaných fragmentů. Jakmile je index vytvořen, nelze změnit počet hlavních fragmentů. Samotná data jsou uložena v již zmíněných komplexních datových strukturách, které jsou serializované jako JSON dokumenty.



Obrázek 1: Vztahy Elasticsearch komponent [3]

[3]Zdroj obrázku: (https://devopsideas.com/)

Elasticsearch má schopnost být bez-schématový, což znamená, že dokumenty mohou být indexovány bez explicitní specifikace, jak zacházet s jednotlivými položkami různého datového typu, které by se mohly v dokumentu vyskytovat. Když je povoleno *dynamické mapování*, Elasticsearch automaticky detekuje datové typy dat a přidává nové položky do indexu. Toto výchozí chování usnadňuje indexaci a prozkoumávání dat. Během indexování Elasticsearch automaticky detekuje a mapuje booleany, desetinné a celočíselné hodnoty a řetězce na odpovídající datové typy v Elasticsearch.

1.2 Dynamické mapování

Když je povoleno *dynamické mapování* (*dynamic mapping*) položek v nastavení mapování indexu, Elasticsearch mapuje některé datové typy pomocí určitých pravidel. Dynamicky mapované datové typy a pravidla jejich mapování jsou uvedeny v tabulce 1. Dynamické mapování řídí parametr `"dynamic"`. Nastavením hodnoty tohoto parametru na hodnotu `"runtime"` se při mapování změní chování Elasticsearch na základě příchozích dokumentů (viz kód 2). Ve výchozím nastavení je hodnota parametru nastavena na `true`. Chceme-li mapování nastavit explicitně, musíme nastavit hodnotu parametru `"dynamic"` na `false` (viz kód 1).

```
1  PUT /my-index-000001
2  {
3    "mappings": {
4      "dynamic": false,
5      "properties": {
6        "age":  { "type": "integer" },
7        "email": { "type": "keyword" },
8        "name": { "type": "text" }
9      }
10   }
11 }
```

Zdrojový kód 1: Explicitní mapování

Tabulka 1: Dynamicky mapované typy

Datový typ v JSON	Datový typ v Elasticsearch "dynamic"	
	`true`	`"runtime"`
`null`	Pole nepřidáno	Pole nepřidáno
`true` nebo `false`	`boolean`	`boolean`
`double`	`float`	`double`
`long`	`long`	`long`
`object`	`object`	Pole nepřidáno
`array`	Závisí na první nenulové hodnotě v poli	Závisí na první nenulové hodnotě v poli
`string` zapisuje datum	`date`	`date`
`string` zapisuje číslo	`float` nebo `long`	`double` nebo `long`
`string` nezapisuje datum ani číslo	`text` s `.keyword` podpolem	`keyword`

Nastavení dynamického parametru na `false` ignoruje nové položky a `"strict"` odmítne dokument, pokud Elasticsearch narazí na položku, jež nemá explicitně nastavené mapování.

```
PUT /index_name
{
  "mappings": {
    "dynamic": "runtime",
    "properties": {
      "field1": {
        "type": "text"
      },
      "field2": {
        "type": "integer"
      }
    }
  }
}
```

Zdrojový kód 2: Nastavení dynamického mapování

1.3 Dynamické šablony

Dynamické šablony (*dynamic template*) umožňují větší kontrolu nad tím, jak Elasticsearch mapuje data nad rámec výchozích pravidel dynamického mapování položek. Dynamické mapování lze povolit nastavením dynamického parametru na hodnotu `true` nebo `"runtime"`. Pomocí dynamických šablon lze definovat

vlastní mapování, která lze použít na dynamicky přidané položky na základě odpovídající podmínky:

- `"match_mapping_type"` se aplikuje na datový typ položky, který Elasticsearch detekuje;
- `"match"` a `"unmatch"` používají vzor pro nalezení shody, který se aplikuje na jméno položky;
- `"path_match"` a `"path_unmatch"` se aplikují na celou tečkovou cestu k položce;
- pokud dynamická šablona nedefinuje `"match_mapping_type"`, `"match"`, nebo `"path_match"`, nenajde shodu s žádnou položkou. Stále je ale možné na šablonu odkazovat jménem v `"dynamic_templates"` sekci *hromadného požadavku* (*bulk request*).

Ve zdrojovém kódu 3 je demonstrováno nalezení položek podle jména. Podmínka `"match"` najde všechny položky začínající nebo končící na `"ip"` a podmínka `"unmatch"` odstraní již nalezené shody, které ji splňují. Takto nalezené položky jsou potom namapovávány na datový typ `"ip"`, který je v Elasticsearch podporován.

```
PUT my-index-000001 {
  "mappings": {
    "dynamic_templates": [
      {
        "ip_fields": {
          "match": ["ip_*", "*_ip"],
          "unmatch": ["one*", "*two"],
          "mapping": {
            "type": "ip"
          }
        }
      }
    ]
  }
}

PUT my-index/_doc/1 {
  "one_ip": "Nenajde shodu",
  "ip_two": "Nenajde shodu",
  "three_ip": "12.12.12.12",
  "ip_four": "13.13.13.13"
}
```

Zdrojový kód 3: Dynamická šablona

1.4 Runtime fields

Elasticsearch umožňuje vytvářet tzv. *runtime položky* (*runtime fields*), což jsou specializované neindexované položky, které jsou vyhodnocovány během dotazování a přistupuje se k nim, jako by to byly indexované položky. Tato funkce je zvláště užitečná pro experimentování s různými typy dat položek nebo skriptů bez nutnosti upravovat základní mapování indexů. Runtime položky nabízejí několik výhod. Spotřebovávají méně místa na disku, protože nejsou indexovány. Navíc poskytují flexibilitu v přístupu k datům a mapování. Na rozdíl od tradičních indexovaných položek lze do dokumentů po procesu zpracování přidávat runtime položky, čímž se zvyšuje pružnost a přizpůsobivost při správě dat.

Runtime položky mohou sloužit jako náhrada mnoha případů použití skriptování v rozhraní *Search API*. Lze například použít runtime položku k načítání hodnot filtrování dokumentů, řazení výsledků a provádění dalších operací. Runtime položky se dobře hodí pro scénáře, kde je potřeba experimentovat s různými datovými typy položek nebo skripty, aniž by se měnilo základní mapování indexů. Hodí se, když je potřeba přidat položky do dokumentů po zpracování dat.

Přes své mnohé výhody je třeba brát v úvahu určité kompromisy. Tyto položky mohou ovlivnit výkon vyhledávání, protože se vyhodnocují během vykonávání dotazu. Rozsah tohoto dopadu závisí na faktorech, jako je složitost výpočtu definovaného v *runtime skriptu* a počet dokumentů zapojených do operace. Runtime skript se spouští při provádění dotazu a generuje hodnoty pro každé pole, které je zahrnuto v dotazu. Pokud je runtime položka často využívána a významně ovlivňuje výkon může být rozumnější ji převést na indexovanou položku. Runtime položku, která neovlivní mapování indexu a je použita pouze v kontexu dotazu, lze definovat přímo ve vyhledávacím dotazu pomocí možnosti `"runtime_mappings"`.

V příkladech s čísly 4 a 5 jsou využité runtime položky. V prvním příkladu k získání a uložení jména dne v týdnu z indexované položky typu `"date"` a ve druhém je runtime položka definovaná ve vyhledávacím dotazu za účelem převedení hodnoty typu `"date"` na `"keyword"`, aby bylo možné ji použít v agregaci `"terms"`.

```
PUT my-index-000001/
{
  "mappings": {
    "runtime": {
      "day_of_week": {
        "type": "keyword",
        "script": {
          "source": "emit(doc['@timestamp'].value.dayOfWeekEnum.
              getDisplayName(TextStyle.FULL, Locale.ROOT))"
        }
      }
    },
    "properties": {
      "@timestamp": {"type": "date"}
    }
  }
}
```

Zdrojový kód 4: Mapování runtime field

```
GET my-index-000001/_search
{
  "runtime_mappings": {
    "day_of_week": {
      "type": "keyword",
      "script": {
        "source": "emit(doc['@timestamp'].value.dayOfWeekEnum.
            getDisplayName(TextStyle.FULL, Locale.ROOT))"
      }
    }
  },
  "aggs": {
    "day_of_week": {
      "terms": {
        "field": "day_of_week"}}}}
```

Zdrojový kód 5: Definice runtime field v dotazu a následné použití k agregaci

1.5 Elasticsearch search API

Elasticsearch API podporuje strukturované vyhledávání, k čemuž slouží jazyk *Query DSL*. Jedno vyhledávání se skládá z jednoho nebo více dotazů, které se zkombinují a pošlou do Elasticsearch. Výsledky vyhledávání se nazývají *hits*. Vyhledávání muže obsahovat další informace, které se použijí pro lepší zpracování dotazů. Vyhledávací API také umožňuje data agregovat.

V textových datech je možné vyhledávat s použitím čistě textových dotazů. Fulltextové dotazy identifikují všechny dokumenty odpovídající dotazovacímu řetězci a vracejí je seřazené podle relevance, tedy podle míry shody s hledanými termíny. Kromě hledání jednotlivých termínů lze provádět frázová hledání, hledání podobnosti a prefixová hledání. K dispozici jsou také automatické návrhy pro doplňování textu.

Elasticsearch podporuje i geografická data, takže je možné např. vyhledávání dokumentů, jež se nachází určitou vzdálenost od sebe nebo umí najít všechny dokumenty, které se nachází v určité oblasti. Posledním typem vyhledávání v Elasticsearch je vektorové vyhledávání, které pomocí algoritmů hledání nejbližšího souseda nebo k-tého nejbližšího souseda, umožňuje hledání podobných vektorů. Lze i použít algoritmy pro zpracování přirozeného jazyka k převedení textu na vektory a provádět tak sémantické vyhledávání. Pracovat s číselnými daty Elasticsearch samozřejmě umí také.

1.6 Query DSL

Query DSL je dotazový jazyk, který je v Elasticsearch používán pro konstrukci dotazů. Je založen na formátu JSON, který používá k vytvoření strukturovaného stromu dotazů. Dotazy jsou tvořeny dvěma typy klauzulí:

- **Listovými vyhledávacími klauzulemi:** Listové klauzule hledají konkrétní hodnotu v konkrétním poli. Tyto klauzule jsou např. `"match"`, `"term"` nebo `"range"` a mohou být použity samy o sobě (viz kód 7).
- **Složenými vyhledávacími klauzulemi:** Složené vyhledávací klauzule slouží k obalování jiných listových nebo složených dotazů. Tyto klauzule se používají logicky ke kombinování několika dotazů do jednoho. Například klauzule `"bool"` nebo `"dis_max"` se často využívají k tomu, aby se spojily různé dotazy. Dále se také používají k úpravě chování dotazů, například klauzule `"constant_score"` slouží k nastavení konstantní váhy pro všechny odpovídající dokumenty (viz kód 6).

Některé typy dotazů mají vyšší výpočetní náročnost z důvodu jejich implementace. Tyto dotazy lze zakázat v nastavení svazku pomocí API nastavením `"search.allow_expensive_queries"` na `false`.

Náročné dotazy jsou:

- Dotazy, které musí provádět lineární skenování.
- Dotazy, které mají vysokou počáteční cenu (`"range"`, `"regex"`, spojování dotazů).
- Dotazy, které můžou mít vysokou cenu na dokument. Např. dotaz s klauzulí `"script_score"`, která umožňuje navrácenému dokumentu přiřadit vlastní skóre.

```
{
  "query": {
    "bool": {
      "should": [
        { "match": { "jmeno_pole": "hledaná_hodnota" } },
        { "term": { "jmeno_pole": "konkrétní_hodnota" } },
        { "range": { "jmeno_pole": { "gte": "min_hodnota", "lte": "
           max_hodnota" } } },
        {
          "bool": {
            "should": [
              { "match": { "jmeno_pole": "hledaná_hodnota" } },
              { "term": { "jmeno_pole": "konkrétní_hodnota" } },
              { "range": { "jmeno_pole": { "gte": "min_hodnota", "lte":
                   "max_hodnota" } } }
            ]
          }
        },
        {
          "dis_max": {
            "queries": [
              { "match": { "jmeno_pole": "hledaná_hodnota" } },
              { "term": { "jmeno_pole": "konkrétní_hodnota" } },
              { "range": { "jmeno_pole": { "gte": "min_hodnota", "lte":
                   "max_hodnota" } } }
            ]
          }
        },
        {
          "constant_score": {
            "filter": { "term": { "jmeno_pole": "hodnota" } }
          }
        }
      ]
    }
  }
}
```

Zdrojový kód 6: Příklad složeného dotazu

```
GET /_search
{
  "query": {
    "match": {
      "text": "konkrétní_hodnota"
    }
  }
}
```

Zdrojový kód 7: Příklad jednoduchého dotazu

1.7 Agregace

Elasticsearch organizuje agregace do tří kategorií:

- *Metriky* (*metric*) jsou agregace, které počítají metriky jako jsou suma nebo průměr z hodnot položek.
- *Kyblíky* (*bucket*) jsou agregace, které seskupují dokumenty do kyblíků na základě hodnot, intervalů nebo jiných kritérií.
- *Potrubí* (*pipeline*) jsou agregace, které přijímají vstup z jiných agregací namísto dokumentů nebo položek.

Agregace se definují pomocí klauzule `"aggs"`, po které následuje libovolné jméno agregace. Agregace je možné libovolně vnořovat do sebe. Metriky se zanořují do kbelíků, protože operace jako suma, průměr atd. se provádějí na dokumentech, které spadají do jednotlivých kyblíků. Potrubí potřebuje mít uvedenou cestu k agregaci, na které chceme provádět další agregaci. Rozsah dokumentů, na kterých jsou agregace prováděny omezuje vyhledávací dotaz s filtry, který je uveden před agregací. Pokud není uveden žádný dotaz, pak se agregace provádějí na všech dokumentech v indexu. Výsledky agregací jsou ukládány do mezipaměti na úrovni fragmentu, pokud ve výsledcích nejsou obsaženy žádné dokumenty tzn. parametr `"size"` ve vyhledávacím dotazu je roven `0` nebo pokud žádný vyhledávací dotaz nebyl uveden.

V příkladu 8 je použita agregace `"date_histogram"` s názvem `"daily_max_temperature"`, která seskupuje dokumenty podle dní a vnořená metrika `"max"` s názvem `"max_temperature"` hledající maximální hodnotu teploty položky `"t"` v jednotlivých kyblících. Následně je pomocí potrubí `"avg_bucket"` se jménem `"avg_daily_max_temperature"` spočítána průměrná denní maximální teplota ze všech denních maximálních teplot.

```
"aggs": {
     "daily_max_temperature": {
       "date_histogram": {
         "field": "date",
         "calendar_interval": "day"
       },
       "aggs": {
         "max_temperature": {
           "max": {
             "field": "t"
           }
         }
       }
     },
     "avg_daily_max_temperature": {
       "avg_bucket": {
         "buckets_path": "daily_max_temperature>max_temperature"
       }
     }
   }
```

Zdrojový kód 8: Příklad agregace

1.8 Vyhledávání v geografických datech

V Elasticsearch jsou k dispozici dva přístupy k práci s geografickými polohami. Prvním z nich je reprezentace *geografických bodů* pomocí datového typu `"geo_point"`, zatímco druhý způsob využívá komplexních *tvarů* definovaných v GeoJSON přes datový typ `"geo_shape"`. Položky typu `"geo_point"` umožňují lokalizaci bodů v dané vzdálenosti od jiného bodu, výpočet vzdáleností mezi dvěma body pro účely třídění či určení relevance a agregaci do mřížky pro zobrazení na mapě. Naopak, položky typu `"geo_shape"` jsou především využívány k filtraci, ať už pro určení překrývajících se tvarů nebo zjištění, zda jeden tvar úplně obsahuje ostatní tvary.

Pro filtrování podle geografických bodů existuje dvojice filtrů: `"geo_bounding_box"` pro určení plochy a `"geo_distance"` k určení vzdálenosti (viz kód 9). Tyto filtry jsou výpočetně náročné a doporučuje se jejich použití omezovat. Nejvýkonnějším geografickým filtrem je `"geo_bounding_box"`, který nepotřebuje načítat všechny hodnoty bodů do paměti.

```
GET /ground2016,ground2017,ground2018/_search
{
  "size": 0,
  "query": {
    "bool": {
      "filter": [
        {
          "geo_distance": {
            "distance": "50km",
            "location": "48.75,-4.01"
          }
        }
      ]
    }
  }
}
```

Zdrojový kód 9: Najde dokumenty, které jsou do vzdálenosti 50 km od `"location"`

1.9 Mezipaměti

Pro zvýšení rychlosti odpovědí na dotazy využívá Elasticsearch několik typů mezipamětí (cache):

- *Stránkovací mezipamět* (*Page cache*) realizuje operační systém. Základní myšlenka stránkovací mezipaměti spočívá v tom, že data jsou po přečtení z disku umístěna do dostupné paměti, aby při dalším čtení byla vrácena z paměti a získání dat nevyžadovalo vyhledávání na disku. Vše je pro aplikaci zcela transparentní. Další čtení stejných dat je potom mnohem rychlejší a na to Elasticsearch spoléhá.

- *Mezipaměť na úrovni fragmentu* (*Shard-level request cache*) ve výchozím nastavení ukládá celé odpovědi na dotazy, které neobsahují dokumenty, ale pouze agregace nebo návrhy (návrhy pro textové vyhledávání). To znamená, že aby byla tato mezipaměť použita, musí být v dotazu nastaven parametr `"size"` na `0`. Toto chování lze změnit nastavením hodnoty parametru v hlavičce dotazu `request_cache` u konkrétního dotazu na `True`.

- *Dotazová mezipaměť* (*Query cache*) uchovává části dotazů, které se několikrát opakují. Konkrétně se jedná o části dotazů zanořené v klauzuli `"filter"`. Každý uzel má jednu dotazovou mezipaměť, která je sdílená mezi všemi fragmenty stejného uzlu. Tato mezipaměť rozhoduje na základě heuristických pravidel, jaké části dotazů se vyplatí uložit, a které ne. Odpovědi na klauzule, které se vyhodnocují rychle, nejsou ukládány vůbec. To jsou např. klauzule `"term"` a `"match_all"`.

- *Položková mezipaměť* (*Field data cache*) uchovává hodnoty nedávno hledaných položek.

1.10 Elasticsearch cluster konfigurace

Každý uzel může mít jednu nebo více rolí. Existují povinné role pro každý svazek a volitelné či specializované role pro specifické typy dat či operací. Uzly se konfigurují prostřednictvím souboru `elasticsearch.yml`, který obsahuje různá nastavení, včetně například `path.data`, který určuje kde se data ukládají na disku. Uzly lze také konfigurovat přes příkazovou řádku či pomocí API. Mezi základní role patří:

- `master:` Uzel, jenž může být zvolen jako *hlavní uzel* (*master node*), který řídí celý svazek.
- `data:` Uzel uchovávající data a provádějící operace související s nimi, např. CRUD, vyhledávání a agregace.
- `ingest:` Uzel schopný aplikovat transformaci dat (*ingest pipeline*) na data před jejich indexací.
- `coordinating:` Uzel směrující požadavky od klientů na ostatní uzly v svazku. Každý uzel implicitně funguje jako *koordinační uzel* (*coordinating node*), pokud nemá explicitně nastavenou roli.
- `remote_cluster_client:` Uzel chovající se jako klient při překračování hranic svazku, umožňující *vyhledáváni napříč svazky*(*crosscluster search*) a replikaci.
- `voting_only:` Uzel účastnící se volby hlavního uzlu, ale nikdy není sám zvolený jako hlavní uzel. Může sloužit jako rozhodčí v případě remízy.

Mezi specializované role patří:

- `ml:` Uzel provádějící úlohy strojového učení a zpracovávající požadavky prostřednictvím API. Vyžaduje rovněž roli `remote_cluster_client` při vyhledávání napříč svazky.
- `transform:` Uzel provádějící transformace a zpracovávající požadavky na transformace. Vyžaduje rovněž roli `remote_cluster_client` při vyhledávání napříč svazky.
- `data_content:` Uzel, jenž uchovává datové záznamy bez časové závislosti, například katalogy produktů nebo archivy článků. Je optimalizován pro rychlé vyhledávání a agregaci.
- `data_hot:` Uzel, jenž uchovává nejnovější a nejčastěji vyhledávaná časově závislá data. Je optimalizován pro rychlé čtení i zápis.

- `data_warm`: Uzel, uchovávající starší časově závislá data, méně často vyhledávaná než data v data_hot.
- `data_cold`: Uzel, který uchovává ještě starší časově závislá data, jež jsou vzácně vyhledávaná.
- `data_frozen`: Uzel, jenž uchovává nejstarší časově závislá data, velmi zřídka vyhledávaná. Je optimalizován pro minimální náklady na úložiště a provoz.

Další důležitá nastavení pro správnou funkci svazku zahrnují `cluster.name` k identifikaci svazku, `node.name` k identifikaci uzlu a možnost `discovery.seed_hosts` pro určení, které uzly mají být použity k nalezení ostatních uzlů v svazku. Některá nastavení jsou specifická pro určité role, jako je například `ml.max_machine_memory_percent` pro uzly pracující se strojovým učením.

2 Příklad použití

Tato kapitola se zabývá tím, jak jsem nastavil a využil Elasticsearch, na jakých datech jsem testoval jeho možnosti a jak jsem měřil jeho výkonost za různých podmínek.

2.1 Data

Data, která jsem si vybral k demonstraci možností Elasticsearch jsou historická meteorologická data ze severu Francie, jež jsou dostupná na stránce `https://meteonet.umr-cnrm.fr/`. Nepoužil jsem všechna data, ale vybral jsem si data získaná z pozemních meteorologických stanic z let 2016 až 2018, protože jejich velikost je dostačující. V datech jsou obsaženy: číslo stanice, zeměpisná šířka, zeměpisná délka, nadmořská výška stanice (m), datum, směr větru (°), rychlost větru (m/s), srážky (mm), vlhkost (%), rosný bod (K), teplota (K) a tlak vzduchu (hPa). Vzorek dat najdete v kódu číslo 10. Tyto data mají celkovou velikost 5 GB a jsou rozděleny do 3 souborů ve formátu csv. Každý soubor obsahuje data získaná v jednom roce.

```
1 number_sta,lat,lon,height_sta,date,dd,ff,precip,hu,td,t,psl,
2 14066001,49.330,-0.430,2.000,20160101
     00:00,210.000,4.400,0.000,91.000,278.450,279.850,
3 14126001,49.150,0.040,125.000,20160101
     00:00,,,0.000,99.000,278.350,278.450,
```

Zdrojový kód 10: Vzorek csv dat

2.2 Nastavení transformace a indexace dat

K transformaci dat a jejich nahrání do Elasticsearch jsem použil nástroj Logstash. Ten mi umožnil specifikovat datové typy jednotlivých položek, formát datumu a transformovat položky `"lat"` a `"lon"` do formátu, který Elasticsearch potřebuje pro indexaci položky typu `"geo_point"`. Původní položky `"lat"` a `"lon"` tak již nebyly potřebné.

Samotné nahrání dat z Logstash do Elasticsearch je díky automatické paralelizaci nahrávání dat, které Logstash zajišťuje, velmi rychlé ve srovnání s nahráváním dat za pomoci elasticsearch python knihovny bez paralelizace.

Data jsem namapoval následovně: Číslo stanice je datového typu `"integer"`, ostatní číselné hodnoty jsou typu `"float"`, datum je přímo typu `"date"` stejného formátu jako ve zdrojových datech, položka `"location"` je typu `"geo_point"` a dynamické mapování je zakázané.

Typ `"date"` umožní vyhledávat dokumenty, jež budou spadat do časových intervalů určených ve vyhledávacích dotazech. Datový typ `"geo_point"` umožní vyhledávání dokumentů, jejichž geografické polohy od sebe budou v dané vzdálenosti nebo budou spadat do dané oblasti.

2.3 Zabezpečení

Zabezpečení komunikace se svazkem je zajištěno pomocí autentizace a SSL, které je možné vypnout, ale není to doporučeno. Já sám jsem jej vypnul a doplatil jsem na to, protože mi byla databáze útočníkem několikrát smazána. Proto minimálně autentizace je nezbytností. Pokud je zabezpečení zapnuto je potom třeba řešit práva uživatelů vytvářením a přidělováním rolí nebo pomocí API klíčů.

2.4 Měření rychlosti vyhodnocování dotazů

Zaměřil jsem se na změření doby vyhodnocování dotazů s použitím mezipamětí a následně porovnání s dobou vyhodnocování bez mezipamětí. Doba vyhodnocení dotazu je zahrnuta v každé odpovědi od svazku, takže v naměřeném čase není zahrnuta odezva sítě, ale pouze doba vyhodnocování dotazu. Dobu vyhodnocení potom počítám jako průměr z naměřených časů. Ke každému průměru je uvedena směrodatná odchylka výběru.

Svazek, na kterém jsem testoval výkon se skládal ze dvou VMware virtualizovaných uzlů s následující systémovou specifikací:

- CPU – Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz,
- RAM – 16GB,
- OS – Linux 5.10.0-22-amd64 (x86_64) Debian GNU/Linux 11 (bullseye).

Použil jsem Elasticsearch verzi 8.11.1 ve výchozím nastavení svazku. V této verzi je ve výchozím nastavení jedna replika na jeden primární fragment. Vybral jsem

několik zajímavých dotazů s náhodnými hodnotami parametrů klauzulí, na kterých jsem výkon testoval. Hodnoty parametrů jednotlivých filtrů byly během měření vybrány náhodně pomocí python knihovny `random` s uniformní pravděpodobností tak, aby odpovídaly rozsahům dat.

První z dotazů je výpočet průměru ze všech maximálních denních teplot v určitém časovém období, ale pouze pro dny, kdy byla rychlost větru vyšší nebo rovna než nějaké minimum a nižší nebo rovna než nějaké maximum a zároveň ze stanic, které byli v určité vzdálenosti od zadaných souřadnic (viz kód 12).

Dotaz se skládá z parametru `"size"`, který udává počet dokumentů, které chceme získat. Je nastavený na `0`, aby se nevrátili žádné dokumenty, protože mě zajímala pouze agregace, abych mohl testovat mezipaměť na úrovni fragmentu.

Následuje klauzule `"query"`, která obsahuje vyhledávací dotaz. Aby bylo možné použít filtrování musí být klauzule `"filter"` zanořená v klauzuli `"bool"`. Klauzule `"filter"` obsahuje seznam filtrů, které vyřadí nežádoucí dokumenty před vyhodnocením výsledné agregace. Dva filtry obsahují klauzuli `"range"`, která filtruje dokumenty podle intervalů hodnot položek `"date"` a `"ff"`. Parametr `"gte"` (větší nebo rovno) je spodní hranice a `"lte"` (menší nebo rovno) je horní hranice intervalu. Posledním filtrem je geografický filtr, jenž vyřadí dokumenty, které nejsou v určité vzdálenosti od zvoleného bodu.

Poslední částí jsou agregace uvnitř klauzule `"aggs"`. První z agregací je agregace `"date_histogram"` se jménem `"daily_max_temperature"`, která seskupí dokumenty podle dní a následuje vnořená agregace `"max"` se jménem `"max_temperature"`, která vrátí maximální denní teplotu. Poslední agregací je `"avg_bucket"` se jménem `"avg_daily_max_temperature"`, která ze všech denních maxim vypočte jejich průměr.

Průměrný čas vyhodnocení z 1000 měření se zapnutými mezipaměťmi (kromě mezipaměti na úrovni fragmentu) je 161 ms. Průměrný čas včetně mezipaměti na úrovni fragmentu z 1000 měření je 131 ms. Průměrný čas bez mezipamětí z 1000 měření je 170 ms. Z výsledků měření lze vyvodit, že na tento dotaz s náhodnými hodnotami parametrů, nemělo použití mezipamětí výrazný efekt.

Elasticsearch před provedením dotazu kalkuluje výpočetní náročnost každého filtru a pomocí toho potom určí nejefektivnější pořadí provedení jednotlivých filtrů. Následně jsem tedy zkusil ovlivnit výpočetní náročnost tak, že jsem generoval náhodné hodnoty parametrů pro každý parametr zvlášť, zatímco hodnoty ostatních parametrů zůstali stejné. Všechny mezipaměti byly zapnuty. Průměrný čas z 1000 měření je 34 ms u náhodného intervalu `"ff"`, 33 ms u náhodného intervalu `"date"` a 83 ms u náhodného `"geo_distance"`. Ve srovnání s náhodnými hodnotami parametrů filtrů (se všemi mezipamětmi) bylo vykonání dotazu s pouze náhodným `"ff"` přibližně 3,85krát rychlejší, s pouze náhodným `"date"` 3,97krát rychlejší a s pouze náhodným `"geo_distance"` 1,58krát rychlejší. Ve vyhodnocování dotazů, které mají společné filtry, tak mezipaměti hrají zásadní roli.

Z výsledků jde i vidět větší náročnost geografických filtrů v porovnání s ostatními filtry. Rozdíl mezi náročnostmi by šel zmírnit indexací zeměpisné délky

a šířky jednotlivě a následně použitím filtru `"geo_bounding_box"` místo `"geo_distance"`. Rozdíl v náročnostech je vidět i ve vizualizační platformě *Kibana*, která k získání dat o náročnostech vyhodnocení jednotlivých částí dotazu používá *Profile API* (viz ob. č. 2). Na obrázku je vidět, že vyhodnocení geografického filtru pro položku `"location"` trvá 27.17 % času vyhodnocení klauzule `"bool"` (*BooleanQuery*), potom následuje datový filtr pro položku `"date"` s 17,74 % a nakonec číselný filtr pro pole `"ff"` s 12,58 %.

Obrázek 2: Vizualizace náročnosti jednotlivých filtrů dotazu MaxTemp

Průměry naměřených hodnot všech ostatních dotazů najdete v tabulce 2 a směrodatné odchylky měření v tabulce 3. Kromě dotazů `GeoQuery` a `PrecipSum` se dotazy liší pouze agregacemi. Všechny dotazy jsou k dispozici v příloze této práce.

Podrobněji se podíváme ještě na dotaz `PrecipSum`, který sečte všechny srážky za každý měsíc v určeném období (viz kód 11). Význam struktury tohoto dotazu je již jistě zřejmá, a tak ji již nebudu popisovat. Tento dotaz ve všech případech, jež jsou uvedeny v tabulce 2 má největší čas vyhodnocení, protože je v něm obsažen pouze jeden filtr. Je-li tedy třeba maximalizovat výkon, uvádění co nejvíce vhodných filtrů je pro nejvyšší rychlost vyhodnocení dotazu nejlepším řešením. Z tabulky je ještě vidět, že u tohoto dotazu mělo použití mezipamětí největší pozitivní dopad na rychlost. Ve srovnání s případem bez použití mezipamětí se rychlost s použitím mezipamětí zvýšila přibližně 1,51krát.

Tabulka 2: Průměrné naměřené časy odpovědí z 1000 měření

	GeoQuery	MaxTemp	MedianTemp	AvgHumidity	PrecipSum
Všechny Cache	110 ms	131 ms	140 ms	144 ms	1308 ms
Bez Cache	111 ms	170 ms	151 ms	153 ms	1981 ms
Bez Request Cache		161 ms	155 ms	221 ms	1966 ms
Náhodné Pouze ff		34 ms	34 ms	35 ms	
Náhodné Pouze Datum		33 ms	32 ms	31 ms	
Náhodný Pouze Geo-filtr		83 ms	83 ms		

Tabulka 3: Směrodatné odchylky měření

	GeoQuery	MaxTemp	MedianTemp	AvgHumidity	PrecipSum
Všechny Cache	78 ms	437 ms	415 ms	416 ms	833 ms
Bez Cache	597 ms	507 ms	80 ms	528 ms	1267 ms
Bez Request Cache		587 ms	556 ms	722 ms	1213 ms
Náhodné Pouze ff		67 ms	67 ms	64 ms	
Náhodné Pouze Datum		14 ms	14 ms	16 ms	
Náhodný Pouze Geo-filtr		38 ms	37 ms		

```
"size": 0,
    "query": {
      "bool": {
        "filter": {
          "range": {
            "date": {
              "gte": "20160101 00:00",
              "lte": "20181231 23:59",
              "format": "yyyyMMdd HH:mm"
            }
          }
        }
      },
      "aggs": {
        "monthly_precipitation": {
          "date_histogram": {
            "field": "date",
            "calendar_interval": "month"
          },
          "aggs": {
            "total_precipitation": {
              "sum": {
                "field": "precip"
              }
            }
          }
        }
      }
    }
```

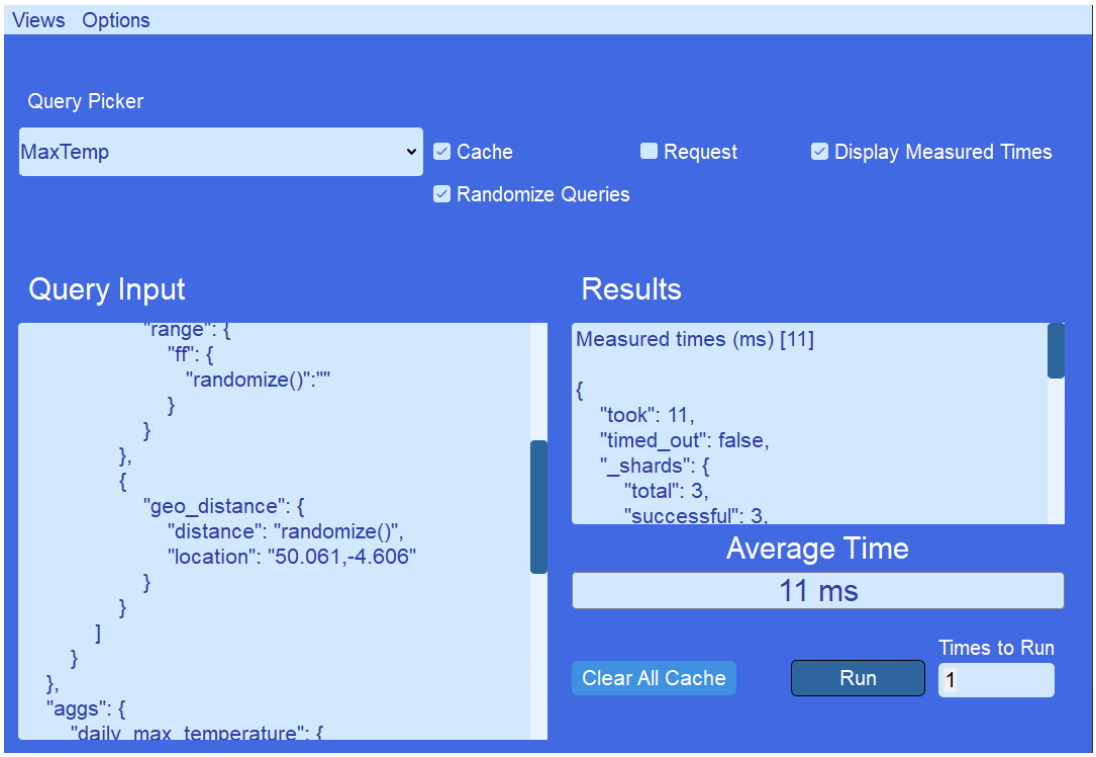
Zdrojový kód 11: Součet měsíčních srážek za období

```
{
   "size": 0,
   "query": {
      "bool": {
         "filter": [
            {
               "range": {
                  "date": {
                     "gte": "20160502 10:52",
                     "lte": "20170529 12:19"
                  }
               }
            },
            {
               "range": {
                  "ff": {
                    "gte": 10.38,
                    "lte": 23.2
                  }
               }
            },
            {
               "geo_distance": {
                  "distance": "50km",
                  "location": "47.725,-0.756"
               }
            }
         ]
      }
   },
   "aggs": {
      "daily_max_temperature": {
         "date_histogram": {
            "field": "date",
            "calendar_interval": "day"
         },
         "aggs": {
            "max_temperature": {
               "max": {
                  "field": "t"
               }
            }
         }
      },
      "avg_daily_max_temperature": {
         "avg_bucket": {
            "buckets_path": "daily_max_temperature>max_temperature"
         }
      }
   }
}
```

Zdrojový kód 12: Průměr teplot

2.5 Aplikace na měření

Pro potřeby měření průměrných časů vyhodnocování dotazů poslaných do Elasticsearch jsem vytvořil jednoduchou grafickou aplikaci v Pythonu. Pro vytvoření grafického rozhraní jsem použil knihovnu PyQt5 a ke komunikaci s elasticsearch svazkem jsem použil knihovnu Elasticsearch DSL. Elasticsearch DSL je vysokoúrovňová knihovna, která abstrahuje nízkoúrovňovou implementaci elasticsearch klienta v Pythonu. K znázornění zatížení svazku jsem použil knihovnu Matplotlib.



Obrázek 3: První obrazovka

V aplikaci jsou dvě obrazovky. První z nich je hlavní obrazovka `Queries` (viz ob. 3), která slouží k měření doby vyhodnocení jednotlivých dotazů poslaných na svazek. Umožňuje vybrat dotaz v menu `Query Picker` z předem připravených dotazů, které jsou načítány z konfiguračního souboru. Uživateli je potom vybraný dotaz zobrazen v levé textové oblasti `Query Input`, kde je možné dotaz před odesláním dále upravovat. Vpravo od této oblasti se nachází další textová oblast `Results`, kde se zobrazí odpověď na poslední odeslaný dotaz, a pokud je zvoleno, tak se i zobrazí všechny naměřené časy. Pod touto oblastí se nachází oblast, kde se zobrazuje průměrný čas zpracování zvoleného počtu odeslaných dotazů na elasticsearch svazek. O něco níže se nachází tlačítko `Clear All Cache`, které po stisknutí odešle požadavek na svazek, který smaže všechny záznamy ze všech mezipamětí. Vpravo od tohoto tlačítka se nachází tlačítko `Run`, které spustí proces odesílaní dotazů na svazek a následně výpočet průměrného času vyhodnocení

dotazů svazkem. Vpravo od tlačítka `Run` se nachází políčko `Times to Run` sloužící k nastavení kolikrát se má dotaz uvedený v `Query Input` oblasti poslat na svazek. Nad oblastí `Results` se nachází zaškrtávací políčka, které slouží k zapnutí nebo vypnutí několika možností:

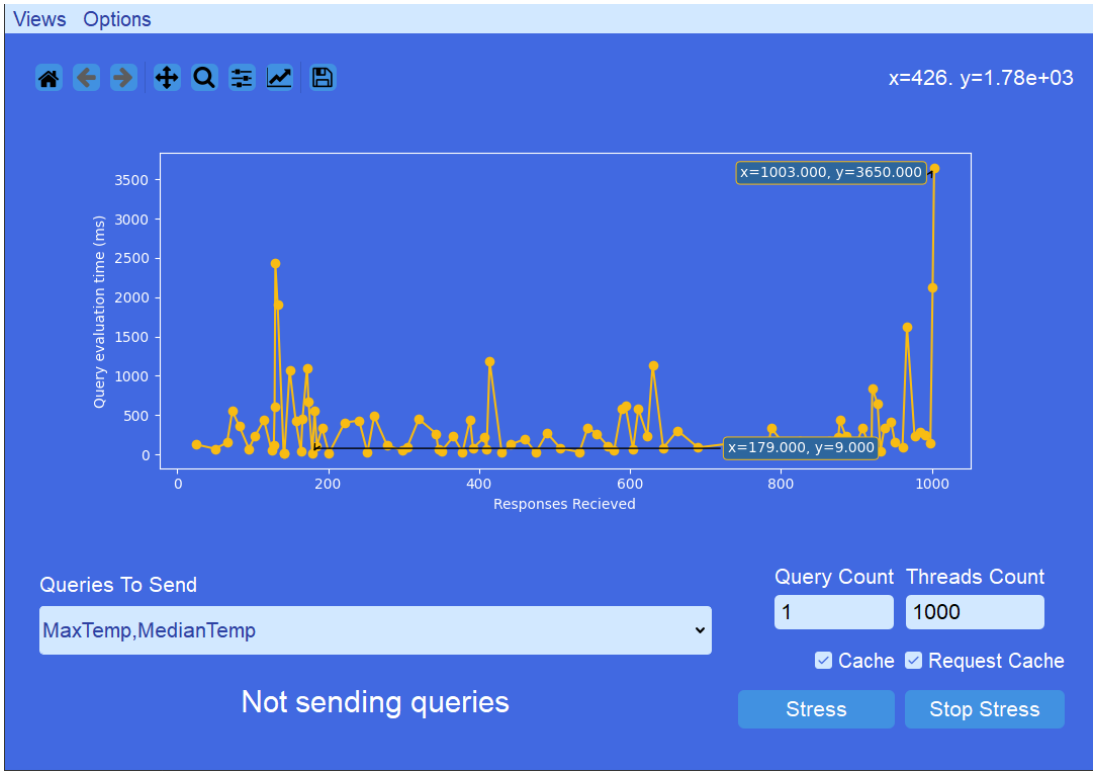
- Povolení nebo vypnutí použití všech mezipamětí svazku při vyhodnocování dotazů (možnost `Cache`).
- Použití mezipaměti na úrovni fragmentu (možnost `Request`).
- Zobrazení všech časů vyhodnocení dotazů v oblasti `Results` (možnost `Display Measured Times`).
- Poslední možností je generování náhodných hodnot všech parametrů dotazu (kromě agregací), jenž je uveden v oblasti `Query Input`, takže před každým odesláním dotazu jsou vygenerovány nové hodnoty.

Je možné generovat náhodně pouze některé hodnoty parametrů a to tak, že v `Query Input` oblasti místo konkrétní hodnoty uvedeme hodnotu `"randomize()"`. K generování náhodných intervalů se neuvádějí parametry `"gte"` a `"lte"`, ale pouze `"randomize()":""` (viz kód č. 13).

V horní části obou obrazovek je hlavní menu obsahující dvě položky. První z nich je `Views`, která slouží k přepínaní mezi obrazovkami a druhá položka `Options` obsahující dvě možnosti pro manipulaci s daty naměřených na první obrazovce. První možnost slouží pro export naměřených dat a druhá možnost k smazání naměřených dat.

```
{
   "range": {
      "ff": {
         "randomize()":""
      }
   }
},
{
   "geo_distance": {
      "distance": "randomize()",
      "location": "49.711,-1.431"
   }
}
```

Zdrojový kód 13: Generace náhodných hodnot



Obrázek 4: Druhá obrazovka

Druhá obrazovka `Performance` (viz ob. 4) slouží k simulaci zátěže svazku paralelním posíláním vybraných dotazů na svazek a následnému grafickému znázornění jejich průměrných časů vyhodnocení v grafu. Hlavním prvkem této obrazovky je graf a nad ním jsou ovládací tlačítka, která slouží k jeho ovládání. Lze jimi měnit pohledy na graf, upravovat osy grafu, měřítko, barvu a uložit graf ve formátu `.png`. Pod grafem se nachází menu `Queries To Send` pro výběr dotazů, které budou odesílány na svazek. Níže se nachází textová indikace, která uživatele informuje, zda se stále odesílají dotazy na svazek. Vpravo od menu a textové indikace se nacházejí políčka `Query Count` udávající počet dotazů, které má každé vlákno odeslat a políčko `Threads Count` udávající počet vláken, jež budou odesílat dotazy na svazek. Pod políčky jsou zaškrtávací okénka `Cache` a `Request Cache`, pomocí kterých může uživatel zapnout nebo vypnout mezipaměti a zvlášť mezipaměť na úrovni fragmentu. Pod těmito políčky se nachází tlačítko `Stress`, které po stisknutí zahájí odesílaní dotazů na svazek a vpravo od něj je tlačítko `Stop Stress` k zastavení odesílání.

Graf se aktualizuje každých 250 ms a znázorňuje průměrný čas zpracování dotazů, na které aplikace obdržela odpověď v uvedené době. Potom co jsou všechny časy znázorněny, tak se v grafu zvýrazní minimální a maximální hodnoty.

Závěr

Elasticsearch nabízí mnoho způsobů, jak data indexovat a vyhledávat. Z výsledků vyplývá, že použití mezipaměti má na rychlost vyhodnocení dotazů potenciálně velký dopad, jsou-li si dotazy navzájem podobné. Jestliže jsou parametry filtrů náhodné, pak není možné využít výrazného zrychlení, které mezipaměti za správných podmínek poskytují.

Je-li třeba optimalizovat rychlost co nejvíce, nesmí se podcenit ani dopad klauzulí náročnějších na vyhodnocení a je dobré využít nějakou podobnou jednodušší klauzuli, pokud je dostupná. Stejně tak je vhodné použít podporovaný datový typ, který nejlépe odpovídá struktuře dat. Pokud i přes všechny optimalizace není výkon vyhovující, pak je možné díky distribuované architektuře systému do svazku přidat další uzel.

V budoucnu by bylo dobré porovnat výkon Elasticsearch s nějakou SQL databází na stejných datech a zjistit, jaké řešení by pro tyto data bylo nejlepší.

Conclusions

Elasticsearch offers many ways to index and search data. The results show that using caching can potentially have a significant impact on query evaluation speed, when queries are similar to each other. If filter parameters are random, then significant acceleration provided by caching under the right conditions is not possible.

If it is necessary to optimize speed as much as possible, the impact of clauses that are more demanding on evaluation should not be underestimated, and it is good to use a simpler similar clause if available. Similarly, it is advisable to use a supported data type that best matches the data structure. If, despite all optimizations, performance is not satisfactory, then it is possible to add another node to the cluster thanks to the distributed architecture of the system.

In the future, it would be good to compare the performance of Elasticsearch with some SQL database on the same data and find out which solution would be best for this data.

A Obsah elektronických dat

Následuje popis elektronických dat práce:

text/
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech (textových) příloh, a všechny soubory potřebné pro bezproblémové vytvoření PDF dokumentu textu (případně v ZIP archivu), tj. zdrojový text textu a příloh, vložené obrázky, apod.

README.txt
: Textový soubor s informacemi o opakovatelném způsobu použití ostatních dat práce. Obsahuje popis instalace klientské aplikace na operační systémy Windows 10 Pro a Debian, popis adresářů a popis nastavení svazku.

aplikace/
: Adresář se soubory spustitelné aplikace. V konfiguračním souboru `config.json` lze změnit dotazy, se kterými aplikace pracuje a lze změnit limity náhodné generace parametrů ve formátu [min, max]. Lze také změnit ip adresu hlavního uzlu svazku a SSL fingerprint, potřebný pro navázání bezpečného spojení se svazkem.

naměřená data/
: Obsahuje soubory s naměřenými daty (doby vyhodnocení dotazů).

nastavení svazku/
: Obsahuje soubor s nastavením mapování indexu `mapování indexu.txt` a soubor `konfigurace Logstash.txt` s nastavením pro Logstash.