Vearch

Release 0.1

Jun 30, 2023

Quick Start

1 Summary **1**
- 1.1 Overall Architecture . . . 2
- 1.2 General Introduction . . . 2
- 1.3 System Features . . . 3

2 Installation and Use **5**
- 2.1 Compile . . . 5
- 2.2 Deploy . . . 6
- 2.3 Use Examples . . . 7

3 Cluster Monitoring **9**
- 3.1 Cluster Status . . . 9
- 3.2 Health Status . . . 9
- 3.3 Port Status . . . 9

4 Database Operation **11**
- 4.1 List Database . . . 11
- 4.2 Create Database . . . 11
- 4.3 View Database . . . 11
- 4.4 Delete Database . . . 11
- 4.5 View Database Space . . . 12

5 Space Operation **13**
- 5.1 Create Space . . . 13
- 5.2 View Space . . . 16
- 5.3 Delete Space . . . 16

6 Doc Opeartion **17**
- 6.1 Single Insertion . . . 17
- 6.2 Batch insertion . . . 18
- 6.3 Update . . . 18
- 6.4 Delete . . . 19
- 6.5 Search . . . 19
- 6.6 ID query . . . 21
- 6.7 Batch query . . . 22
- 6.8 Multi vector query . . . 22

7 Effect Evaluation 23

8 Cluster experiments 27

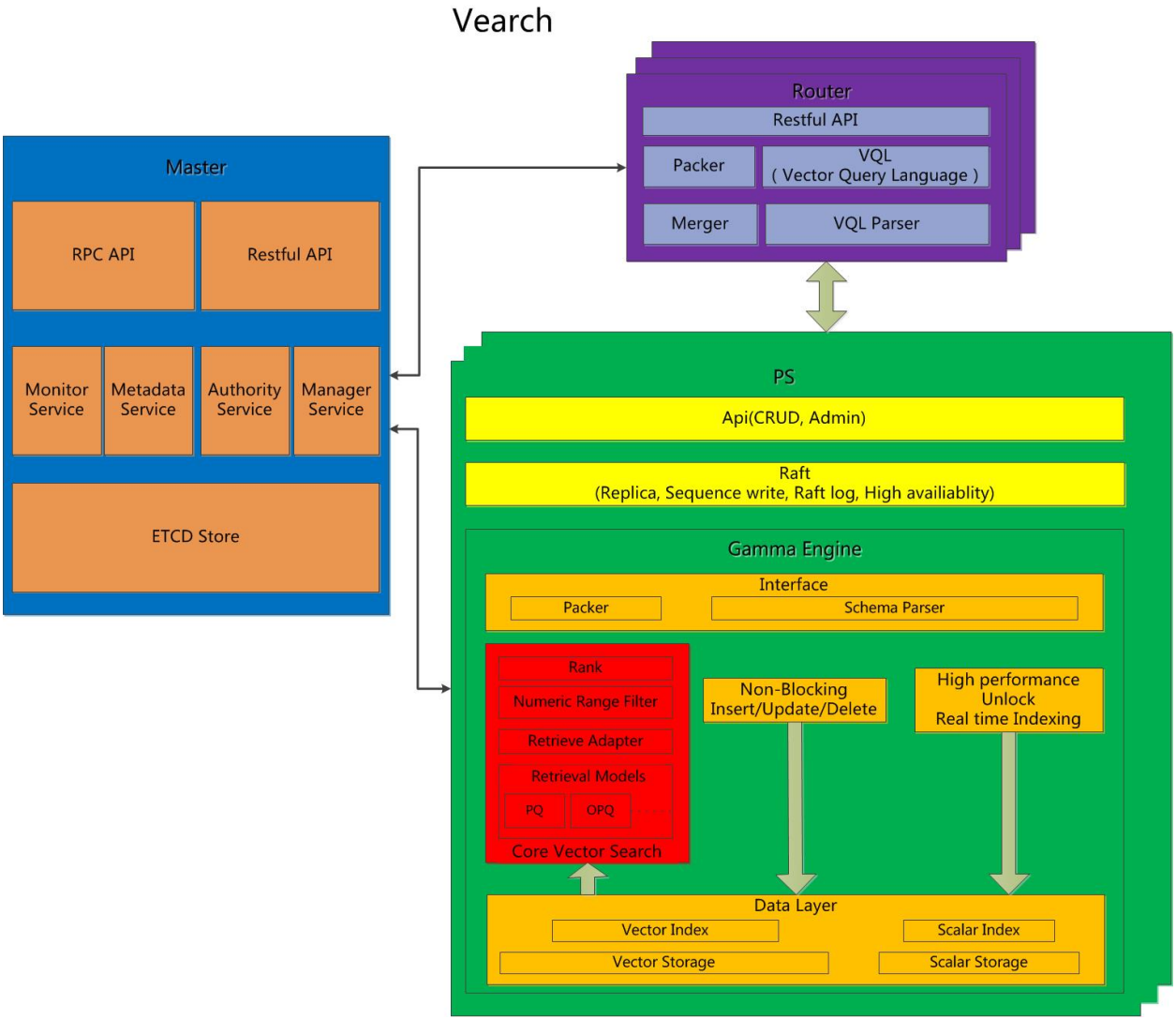
9 Common Problem 29

CHAPTER 1

Summary

Vearch is a scalable distributed system for efficient similarity search of deep learning vectors.

1.1 Overall Architecture



Data Model: space, documents, vectors, scalars

Components: MasterRoutermPartitionServer

Master: Responsible for schema mananagement, cluster-level metadata, and resource coordination.

Router: Provides RESTful API: create , delete search and update ; request routing, and result merging.

PartitionServer(PS): Hosts document partitions with raft-based replication. Gamma is the core vector search engine. It provides the ability of storing, indexing and retrieving the vectors and scalars.

1.2 General Introduction

1. One document one vector.
2. One document multiple vectors.
3. One document has multiple data sources and vectors.

4. Numerical field filtration
5. Batch operations to support addition and search.

1.3 System Features

1. Gamma engine implemented by C++ guarantees fast detection of vectors.
2. Supporting Interior Product and L2 Method to Calculate Vector Distance.
3. Supporting memory and disk data storage, supporting super-large data scale.
4. Data multi copy storage based on raft protocol.

CHAPTER 2

Installation and Use

2.1 Compile

Environmental dependence

1. Linux system(recommend CentOS 7.2 or more), supporting cmake and make commands.
2. Go version 1.11.2 or more
3. Gcc version 5 or more
4. Faiss

Compile

- download source code: git clone https://github.com/vearch/vearch.git (Follow-up use of $vearch to represent the absolute path of the vearch directory)
- compile gamma
    1. `cd $vearch/engine/gamma/src`
    2. `mkdir build && cd build`
    3. `export Faiss_HOME=faiss`
    4. `cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=$vearch/ps/engine/gammacb/lib ..`
    5. `make && make install`
- compile vearch
    1. `cd $vearch`
    2. `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$vearch/ps/engine/gammacb/lib/lib`
    3. `export Faiss_HOME=faiss`
    4. `go build -a --tags=vector -o vearch`

generate `vearch`file compile success

2.2 Deploy

stand-alone mode:

- generate configuration file conf.toml

```
[global]
    # the name will validate join cluster by same name
    name = "vearch"
    # you data save to disk path ,If you are in a production environment, You'd
↪better set absolute paths
    data = ["datas/"]
    # log path , If you are in a production environment, You'd better set absolute
↪paths
    log = "logs/"
    # default log type for any model
    level = "debug"
    # master <-> ps <-> router will use this key to send or receive data
    signkey = "vearch"
    skip_auth = true

# if you are master you'd better set all config for router and ps and router and ps
↪use default config it so cool
[[masters]]
    # name machine name for cluster
    name = "m1"
    # ip or domain
    address = "127.0.0.1"
    # api port for http server
    api_port = 8817
    # port for etcd server
    etcd_port = 2378
    # listen_peer_urls List of comma separated URLs to listen on for peer traffic.
    # advertise_peer_urls List of this member's peer URLs to advertise to the rest of
↪the cluster. The URLs needed to be a comma-separated list.
    etcd_peer_port = 2390
    # List of this member's client URLs to advertise to the public.
    # The URLs needed to be a comma-separated list.
    # advertise_client_urls AND listen_client_urls
    etcd_client_port = 2370

[router]
    # port for server
    port = 9001

[ps]
    # port for server
    rpc_port = 8081
    # raft config begin
    raft_heartbeat_port = 8898
    raft_replicate_port = 8899
    heartbeat-interval = 200 #ms
    raft_retain_logs = 10000
    raft_replica_concurrency = 1
    raft_snap_concurrency = 1
```

- start

```
./vearch -conf conf.toml
```

2.3 Use Examples

CHAPTER 3

Cluster Monitoring

http://master_server is the master service.

3.1 Cluster Status

```
curl -XGET http://master_server/_cluster/stats
```

3.2 Health Status

```
curl -XGET http://master_server/_cluster/health
```

3.3 Port Status

```
curl -XGET http://master_server/list/server
```

CHAPTER 4

Database Operation

http://master_server is the master service, $db_name is the name of the created database.

4.1 List Database

```
curl -XGET http://master_server/list/db
```

4.2 Create Database

```
curl -XPUT -H "content-type:application/json" -d '{
  "name": "db_name"
}' http://master_server/db/_create
```

4.3 View Database

```
curl -XGET http://master_server/db/$db_name
```

4.4 Delete Database

```
curl -XDELETE http://master_server/db/$db_name
```

Cannot delete if there is a table space under the datebase.

4.5 View Database Space

```
curl -XGET http://master_server/list/space?db=$db_name
```

CHAPTER 5

Space Operation

http://master_server is the master service, $db_name is the name of the created database, $space_name is the name of the created tablespace.

5.1 Create Space

```
curl -XPUT -H "content-type: application/json" -d'
{
    "name": "space1",
    "partition_num": 1,
    "replica_num": 1,
    "engine": {
        "name": "gamma",
        "index_size": 70000,
        "max_size": 10000000,
        "nprobe": 10,
        "metric_type": "InnerProduct",
        "ncentroids": 256,
        "nsubvector": 32
    },
    "properties": {
        "field1": {
            "type": "keyword"
        },
        "field2": {
            "type": "integer"
        },
        "field3": {
            "type": "float",
                "index": "true"
        },
        "field4": {
            "type": "string",
```

(continues on next page)

(continued from previous page)

```
            "array": true,
            "index": "true"
        },
        "field5": {
            "type": "integer",
            "array": false,
            "index": "true"
        },
        "field6": {
            "type": "vector",
            "dimension": 128
        },
        "field7": {
            "type": "vector",
            "dimension": 256,
            "store_type": "MemoryOnly",
            "store_param": {
            }
        }
    }
}
' http://master_server/space/$db_name/_create
```

Parameter description:

field name	field description	field type	must	remarks
name	space name	string	true	
partition_num	partition number	int	true	
replica_num	replica number	int	true	
engine	engine config	json	true	
properties	schema config	json	true	define space field

1Space name not be empty, do not start with numbers or underscores, try not to use special characters, etc.

2partition_num: Specify the number of tablespace data fragments. Different fragments can be distributed on different machines to avoid the resource limitation of a single machine.

3replica_num: The number of copies is recommended to be set to 3, which means that each piece of data has two backups to ensure high availability of data.

engine config:

field name	field description	field type	must	remarks
name	engine name	string	true	currently fixed to gamma
index_size	slice index threshold	int	false	
max_size	maximum number of records in segments	int	false	
nprobe	number of cable bins	int	false	default 10
metric_type	calculation method	string	false	InnerProduct and L2, default L2
ncentroids		int	false	default 256
nsubvector	PQ disassembler vector size	int	false	default 64, must be a multiple of 4

1. index_size: Specify the number of records in each partition to start index creation. If not specified, no index will be created.

2. max_size: Specify the maximum number of records stored in each partition to prevent excessive server memory consumption.
3. nprobe: The number of buckets to search in the index cannot be greater than the value of ncentroids.
4. metric_type: Specify the calculation method, inner product or Euclidean distance.
5. ncentroids: Specifies the number of buckets for indexing.
6. nsubvector: PQ disassembler vector size.

properties config:

1. There are four types (that is, the value of type) supported by the field defined by the table space structure: keyword, integer, float, vector (keyword is equivalent to string).
2. The keyword type fields support index and array attributes. Index defines whether to create an index, and array specifies whether to allow multiple values.
3. Integer, float type fields support the index attribute, and the fields with index set to true support the use of numeric range filtering queries.
4. Vector type fields are feature fields. Multiple feature fields are supported in a table space. The attributes supported by vector type fields are as follows:

field name	field description	field type	must	remarks
dimension	feature dimension	int	true	Value is an integral multiple of the above nsubvector value
store_type	feature storage type	string	false	support Mmap and RocksDB, default Mmap
store_param	storage parameter settings	json	false	set the memory size of data
model_id	feature plug-in model	string	false	Specify when using the feature plug-in service

5. dimension: define that type is the field of vector, and specify the dimension size of the feature.
6. store_type: raw vector storage type, there are the following three options

“MemoryOnly”: Vectors are stored in the memory, and the amount of stored vectors is limited by the memory. It is suitable for scenarios where the amount of vectors on a single machine is not large (10 millions) and high performance requirements

“RocksDB”: Vectors are stored in RockDB (disk), and the amount of stored vectors is limited by the size of the disk. It is suitable for scenarios where the amount of vectors on a single machine is huge (above 100 millions) and performance requirements are not high.

“Mmap”: Vectors are stored in the disk file, and the amount of stored vectors is limited by the size of the disk. It is suitable for scenarios where the amount of vectors on a single machine is huge (above 100 millions) and performance requirements are not high.

7. store_param: storage parameters of different store_type, it contains the following two sub-parameters

cache_size: interge type, the unit is M bytes, the default is 1024. When store_type=”RocksDB”, it indicates the read buffer size of RocksDB. The larger the value, the better the performance of reading vector. Generally set 1024, 2048, 4096 and 6144; when store_type=”Mmap”, it indicates the size of the write buffer , Don’t need to be too big, generally 512, 1024 or 2048 will do; store_type=”MemoryOnly”, it is useless.

compress: bool type, default false. True means to compress the original vector, generally the original vector will be compressed to 50% of the original, which can save memory and disk; false means no compression.

5.2 View Space

```
curl -XGET http://master_server/space/$db_name/$space_name
```

5.3 Delete Space

```
curl -XDELETE http://master_server/space/$db_name/$space_name
```

CHAPTER 6

Doc Opeartion

http://router_server is the router service, $db_name is the name of the created database, $space_name is the name of the created space, $ID is the unique ID of the data record.

6.1 Single Insertion

Insert without a unique ID

```
curl -XPOST -H "content-type: application/json"  -d'
{
    "field1": "value1",
    "field2": "value2",
    "field3": {
        "feature": [0.1, 0.2]
    }
}
' http://router_server/$db_name/$space_name
```

field1 and field2 are scalar field and field3 is feature field. All field names, value types, and table structures are consistent

The return value format is as follows:

```
{
    "_index": "db1",
    "_type": "space1",
    "_id": "AW5J1lNmJG6WbbCkHrFW",
    "status": 201,
    "_version": 1,
    "_shards": {
        "total": 0,
        "successful": 1,
        "failed": 0
```

(continues on next page)

(continued from previous page)

```
    },
    "result": "created",
    "_seq_no": 1,
    "_primary_term": 1
}
```

Among them, _index is the name of the database, _type is the name of the tablespace. ID is the unique identification of the record generated by the server, which can be specified by the user. The unique identification needs to be used for data modification and deletion.

Specify a unique ID when inserting

```
curl -XPOST -H "content-type: application/json"  -d'
{
    "field1": "value1",
    "field2": "value2",
    "field3": {
        "feature": [0.1, 0.2]
    }
}

' http://router_server/$db_name/$space_name/$id
```

$id is the unique ID generated by the server with the specified value when inserting data. The $id value cannot use special characters such as URL path. Overwrite if the unique record already exists in the library.

6.2 Batch insertion

```
curl -H "content-type: application/json" -XPOST -d'
{"index": {"_id": "v1"}}\n
{"field1": "value", "field2": {"feature": []}}\n
{"index": {"_id": "v2"}}\n
{"field1": "value", "field2": {"feature": []}}\n
' http://router_server/$db_name/$space_name/_bulk
```

like json format, {“index”: {“_id”: “v1”}} specify the record id, {“field1”: “value”, “field2”: {“feature”: []}} specify inserted dataevery line is json string.

6.3 Update

Unique ID must be specified when updating

```
curl -H "content-type: application/json" -XPOST -d'
{
    "doc": {
        "field1": 32
    }
}
' http://router_server/$db_name/$space_name/$id/_update
```

The unique $id is specified in the request path. The field1 is the field to be modified. The modification of the vector field uses the method of inserting the specified $id to update the data coverage.

6.4 Delete

Delete data according to unique ID

```
curl -XDELETE http://router_server/$db_name/$space_name/$id
```

Delete data according to query filtering results

```
curl -H "content-type: application/json" -XPOST -d'
{
    "query": {
        "sum": [{}]
    }
}
' http://router_server/$db_name/$space_name/_delete_by_query
```

Batch delete according to ID

```
curl -H "content-type: application/json" -XPOST -d'
{"delete": {"_id": "v1"}}
{"delete": {"_id": "v2"}}
{"delete": {"_id": "v3"}}
' http://router_server/$db_name/$space_name/_bulk
```

See the following for query syntax

6.5 Search

Query example

```
curl -H "content-type: application/json" -XPOST -d'
{
    "query": {
        "sum": [{
            "field": "field_name",
            "feature": [0.1, 0.2, 0.3, 0.4, 0.5],
            "min_score": 0.9,
            "boost": 0.5
        }],
        "filter": [{
            "range": {
                "field_name": {
                    "gte": 160,
                    "lte": 180
                }
            }
        },
        {
             "term": {
                 "field_name": ["100", "200", "300"],
                 "operator": "or"
             }
        }]
    },
    "direct_search_type": 0,
```

(continues on next page)

(continued from previous page)

```
    "quick": false,
    "vector_value": false,
    "online_log_level": "debug",
    "size": 10
}
' http://router_server/$db_name/$space_name/_search
```

The overall JSON structure of query parameters is as follows:

```
{
    "query": {
        "sum": [],
        "filter": []
    },
    "direct_search_type": 0,
    "quick": false,
    "vector_value": false,
    "online_log_level": "debug",
    "size": 10
}
```

Parameter Description:

field name	field type	must	remarks
sum	json array	true	query feature
filter	json array	false	query criteria filtering: numeric filtering + label filtering
direct_search_type	int	false	default 0
quick	bool	false	default false
vector_value	bool	false	default false
online_log_level	string	false	set debug, Turn on Printing debug log
size	int	false	number of returned results

- sum json structure elucidation:

```
"sum": [{
           "field": "field_name",
           "feature": [0.1, 0.2, 0.3, 0.4, 0.5],
           "min_score": 0.9,
           "boost": 0.5
       }]
```

(1) sum: Support multiple (including multiple feature fields when defining table structure correspondingly).

(2) field: Specifies the name of the feature field when the table is created.

(3) feature: Transfer feature, dimension must be the same when defining table structure

(4) min_score: Specify the minimum score of the returned result, the similarity between the two vector calculation results is between 0-1, min_score can specify the minimum score of the returned result, and max_score can specify the maximum score. For example, set "min_score": 0.8, "max_score": 0.95 to filter the result of 0.8 <= score <= 0.95. At the same time, another way of score filtering is to use the combination of "symbol": ">=", "value": 0.9. The value types supported by symbol include: >, >=, < and <= four kinds, and the values of value, min_score and max_score are between 0 and 1.

(5) boost: Specify the weight of similarity. For example, if the similarity score of two vectors is 0.7 and boost is set to 0.5, the returned result will multiply the score 0.7 * 0.5, which is 0.35.

- filter json structure elucidation:

```
"filter": [
             {
                 "range": {
                     "field_name": {
                          "gte": 160,
                          "lte": 180
                     }
                 }
             },
             {
                 "term": {
                     "field_name": ["100", "200", "300"],
                     "operator": "or"
                 }
             }
          ]
```

(1) filter: Multiple conditions are supported. Multiple conditions are intersecting.

(2) range: Specify to use the numeric field integer / float filtering, the file name is the numeric field name, gte and lte specify the range, lte is less than or equal to, gte is greater than or equal to, if equivalent filtering is used, lte and gte settings are the same value. The above example shows that the query field_name field is greater than or equal to 160 but less than or equal to 180.

(3) term: With label filtering, field_name is a defined label field, which allows multiple value filtering. You can intersect "operator": "or", merge: "operator": "and". The above example indicates that the query field name segment value is "100", "200" or "300".

- direct_search_type: Specify the query type. 0 means to use index if the feature has been created, and violent search if it has not been created; - 1 means to use index only for search, and 1 means not to use index only for violent search. The default value is 0.
- quick: By default, the PQ recall vector is calculated and refined in the search results. In order to speed up the processing speed of the server to true, only recall can be specified, and no calculation and refined.
- vector_value: In order to reduce the network overhead, the search results contain only scalar information fields without feature data by default, and set to true to specify that the returned results contain the original feature data.
- online_log_level: Set "debug" to specify to print more detailed logs on the server, which is convenient for troubleshooting in the development and test phase.
- size: Specifies the maximum number of results to return. if request address like http://router_server/\protect\T1\textdollardb_name/\protect\T1\textdollarspace_name/_search?size=20, use the size value specified in the URL first.

6.6 ID query

```
curl -XGET http://router_server/$db_name/$space_name/$id
```

6.7 Batch query

```
curl -H "content-type: application/json" -XPOST -d'
{
    "query": {
        "sum": [{
            "field": "vector_field_name",
            "feature": [0.1, 0.2]
        }]
    }
}
' http://router_server/$db_name/$space_name/_msearch
```

The difference between batch query and single query is that the batch features are spliced into a feature array in order, and the background service will split according to the feature dimension when defining the table space structure. For example, define 10-dimensional feature fields, query 50 features in batches, and splice features into a 500 dimensional array in order to assign them to feature parameters. The request suffix uses "_msearch".

6.8 Multi vector query

The definition of tablespace supports multiple feature fields, so the query can support the features of corresponding data. Take two vectors per record as an example: define table structure fields

```
{
    "field1": {
        "type": "vector",
        "dimension": 128
    },
    "field2": {
        "type": "vector",
        "dimension": 256
    }
}
```

Field1 and field2 are vector fields, and two vectors can be specified for search criteria during query:

```
{
    "query": {
        "sum": [{
            "field": "filed1",
            "feature": [0.1, 0.2, 0.3, 0.4, 0.5],
            "min_score": 0.9
        },
        {
            "field": "filed2",
            "feature": [0.8, 0.9],
            "min_score": 0.8
        }]
    }
}
```

The results of field1 and field2 are intersected, and other parameters and request addresses are consistent with those of ordinary queries.

CHAPTER 7

Effect Evaluation

Benchmarks

This document shows the experiments we do and the results we get. Here we do two series of experiments. First, we experiment on a single node to show the recalls of the modified IVFPQ model which is based on faiss. Second, we do experiments with Vearch cluster.

We evaluate methods with the recall at k performance measure, which is the proportion of results that contain the ground truth nearest neighbor when returning the top k candidates (for k {1,10,100}). And we use Euclidean neighbors as ground truth.

Note that the numbers (especially QPS) change slightly due to changes in the implementation, different machines, etc.

Getting data

We do experiments on two kind of features. One is 128-dimensional SIFT feature, the other is 512-dimensional VGG feature.

Getting SIFT1M

To run it, please download the ANN_SIFT1M dataset from

http://corpus-texmex.irisa.fr/

and unzip it to the subdirectory sift1M.

Getting VGG1M and VGG10M

We get 1 million and other 10 million data and then use deep-learning model vgg to get their features.

Getting VGG100M , VGG500M and VGG1B

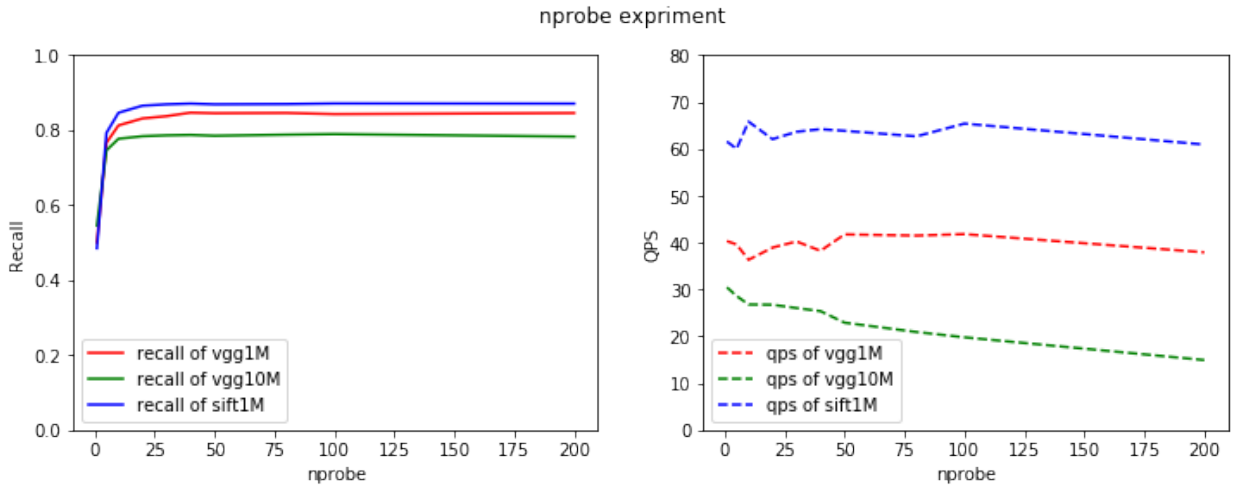
We collect billions of data and use deep-learning model vgg to get their features for cluster experiments.

Nprobe experiments

We do experiments on SIFT1M, VGG1M and VGG10M. In this experiment, nprobe {1,5,10,20,30,40,50,80,100,200}. At the same time, we set the ncentroids as 256 and the nbytes as 32.

We use recall at 1 to show the result.

Result

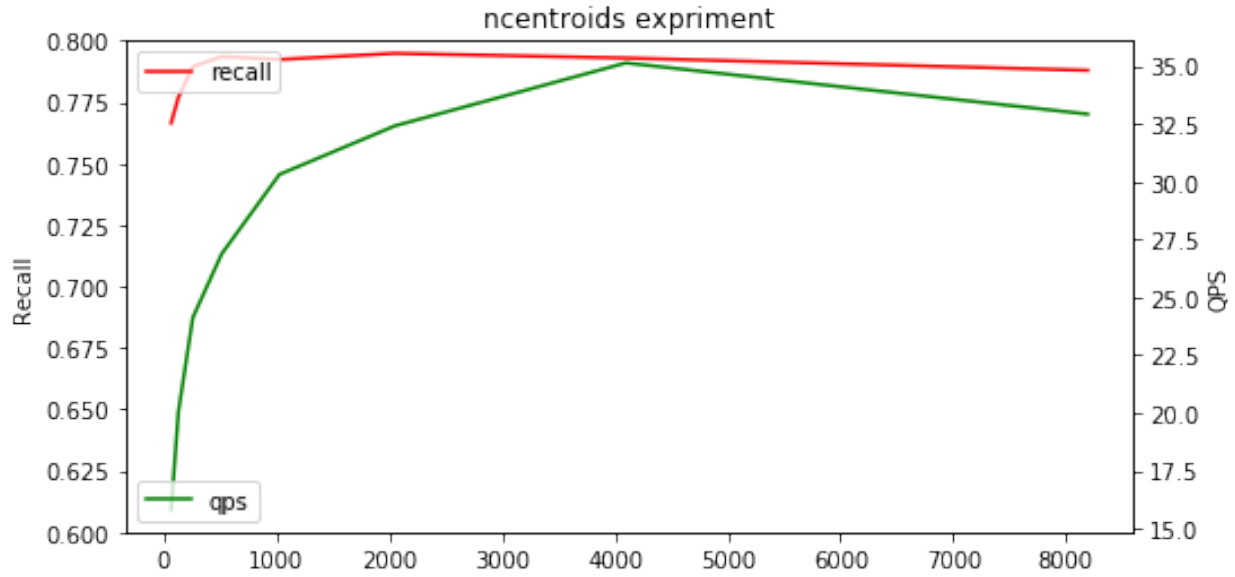


As we can see, when nprobe exceeds 25, there is no obvious change of recalls. Also, when nprobe get larger,only QPS of vgg10M get smaller, QPS of vgg1M and QPS of sift1M basically have no changes.

Ncentroids experiments

We do experiment on VGG10M. The number of centroid {64,128,256,512,1024,2048,4096,8192} and we set nprobe as 50 considering the number of centroid becomes very large. Here we also set nbytes as 32. We use recall at 1 to show the result.

Result

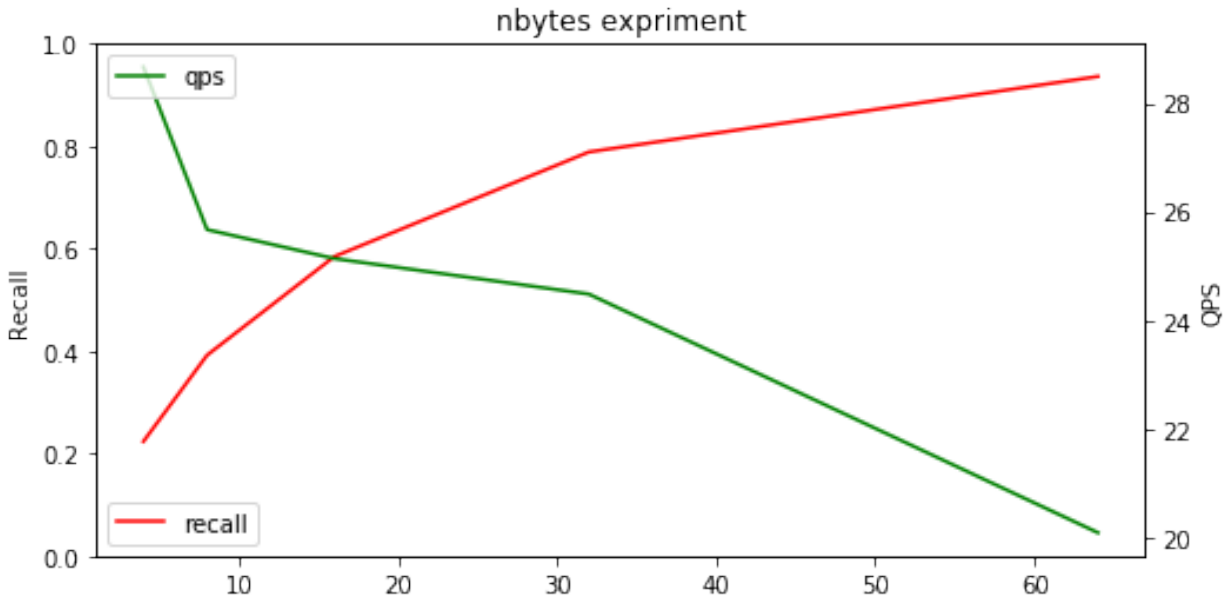


As we can see, there is no obvious change of recalls when the number of centroid get larger. But the QPS become higher and higher as the number of centroid grows.

Nbytes experiments

We do experiment on VGG10M. The number of byte {4,8,16,32,64}. We set ncentroids as 256 and nprobe as 50. We use recall at 1 to show the result.

Result



As we can see, when the number of byte grows, the recall get higher and higher, but the QPS drops obviously.

Experiments with faiss

We do experiments on SIFT1M, VGG1M and VGG10M to compare the recalls with faiss. We use some algorithm implemented with faiss and we use Vearch to represent our algorithm.

Models

Here we show the parameters we set for used models. When the parameters in the table are empty, there are no corresponding parameters in the models. And the parameters of links, efSearch and efConstruction are defined in faiss of hnsw.

model	ncentroids	nprobe	bytes of SIFT	bytes of VGG	links	efSearch	efConstruction
pq |			32	64			
ivfpq |256		20	32	64			
imipq	2^(2*10)	2048	32	64			
opq+pq			32	64			
hnsw					32	64	40
ivfhnsw	256	20			32	64	40
Vearch	256	20	32	64			

Result

recalls of SIFT1M:

model	recall@1	recall@10	recall@100
pq	0.6274	0.9829	0.9999
ivfpq	0.6167	0.9797	0.9960
imipq	0.6595	0.9775	0.9841
opq+pq	0.6250	0.9821	1.0000
hnsw	0.9792	0.9867	0.9867
ivfhnsw	0.9888	0.9961	0.9961
Vearch	0.8649	0.9721	0.9722

recalls of VGG1M :

model	recall@1	recall@10	recall@100
pq	0.5079	0.8922	0.9930
ivfpq	0.4985	0.8792	0.9704
imipq	0.5077	0.8618	0.9248
opq+pq	0.5213	0.9105	0.9975
hnsw	0.9496	0.9550	0.9551
ivfhnsw	0.9690	0.9744	0.9745
Vearch	0.9536	0.9582	0.9585

recalls of VGG10M :

model	recall@1	recall@10	recall@100
pq	0.5842	0.8980	0.9888
ivfpq	0.5913	0.8896	0.9748
imipq	0.5925	0.8878	0.9570
opq+pq	0.6126	0.9160	0.9944
hnsw	0.8877	0.9069	0.9074
ivfhnsw	0.9638	0.9839	0.9843
Vearch	0.9272	0.9464	0.9468

CHAPTER 8

Cluster experiments

First, we do experiments by searching on cluster only with vgg features. Then, we experiment with the vgg features and filter the search using an integer field to compare the time consumed and QPS with the vgg features only. In the following section, we use searching with filter or without filter to specify the experiment method mentioned earlier. For different size of experiment data, we use different Vearch cluster. We use 3 masters, 3 routers and 5 partition services for VGG100M. For VGG500M, we use the same size of master and router with VGG100M but 24 partition services. We use 3 masters, 6 routers and 48 partition services to deal with the VGG1B.

Result

The growth shape of QPS is more like inverted J-shaped curve which means the growth of QPS basically have no obvious change when average latency exceed one certain number.

CHAPTER 9

Common Problem

1. Vearch's vector search engine gamma is based on faiss. Vearch may not compile successfully when the version of faiss is greatly changed and incompatible with the historical version.