MEP 33 -- Support Range Search

Current state: ["Approved"]

ISSUE: #17599

PRs:

Keywords: search, range search, radius, range filter

Released:

Summary(required)

By now, the behavior of "search" in Milvus is returning TOPK most similar sorted results for each queried vector.

This MEP is about to realize a functionality named "range search". User specifies a range scope -- including radius and range filter (optional), Milvus does "range search", and also returns TOPK sorted results with distance falling in this range scope.

- param "radius" MUST be set, param "range filter" is optional
- falling in range scope means

Metric type	Behavior
IP	radius < distance <= range_filter
L2 and other metric types	range_filter <= distance < radius

Motivation(required)

Many users request the "range search" functionality. With this capability, user can

- get results with distance falling in a range scope
- get search results more than 16384

Public Interfaces(optional)

- No interface change in Milvus and all SDKs

  We reuse the SDK interface **Search()** to do "range search". Only add 2 more parameters "radius" and "range_filter" into params.

  When param "radius" is specified, Milvus does range search; otherwise, Milvus does search.

  As shown in the following figure, set **"range_filter": 1.0, "radius": 2.0** in search_ params.params.

```
  default_index = {"index_type": "HNSW", "params":{"M": 48, "efConstruction": 500}, "metric_type": "L2"}
  collection.create_index("float_vector", default_index)
  collection.load()
  search_params = {"metric_type": "L2", "params": {"ef": 32, "range_filter": 1.0, "radius": 2.0}}
  res = collection.search(vectors[:nq], "float_vector", search_params, limit, "int64 >= 0")
```

- 3 new interfaces in knowhere

```
  // range search parameter legacy check
  virtual bool
  CheckRangeSearch(Config& cfg, const IndexType type, const IndexMode mode);

  // range search
  virtual DatasetPtr
  QueryByRange(const DatasetPtr& dataset, const Config& config, const faiss::BitsetView bitset);

  // brute force range search
  static DatasetPtr
  BruteForce::RangeSearch(const DatasetPtr base_dataset, const DatasetPtr query_dataset, const Config& config,
const faiss::BitsetView bitset);
```

Design Details(required)

Knowhere

Index types and metric types supporting range search are listed below:

	IP	L2	HAMMING	JACCARD	TANIMOTO	SUBSTRUCTURE	SUPERSTRUCTURE
BIN_IDMAP			☑	☑	☑		
BIN_IVF_FLAT			☑	☑	☑		
IDMAP	☑	☑					
IVF_FLAT	☑	☑					
IVF_PQ	☑	☑					
IVF_SQ8	☑	☑					
HNSW	☑	☑					
ANNOY							
DISKANN	☑	☑					

If call range search API with unsupported index types or unsupported metric types, exception will be thrown out.

In Knowhere, two new parameters "radius" and "range_filter" are passed in via config, and range search will return **all un-sorted** results with distance falling in this range scope.

Metric type	Behavior
IP	radius **<** distance **<=** range_filter
L2 or other metric types	range_filter **<=** distance **<** radius

Knowhere run range search in 2 steps:

1. pass param "radius" to thirdparty to call their range search APIs, and get result
2. if param "range_filter " is set, filter above result and return; if not, return above result directly

We add 3 new APIs CheckRangeSearch(), QueryByRange() and BruteForce::RangeSearch() to support range search.

1. **CheckRangeSearch()**

   This API is used to do range search parameter legacy check. It will throw exception when parameter legacy check fail.

   The valid scope for "radius" is defined as: -1.0 <= radius <= float_max

metric type	range	similar	not similar
L2	[0, inf)	0	inf
IP	[-1, 1]	1	-1
jaccard	[0, 1]	0	1
tanimoto	[0, inf)	0	inf
hamming	[0, n]	0	n

2. QueryByRange()

This API returns all unsorted results with distance falling in the specified range scope.

PROTO	`virtual DatasetPtr` `QueryByRange(const DatasetPtr& dataset, const Config& config, const faiss::BitsetView bitset)`

INPUT	Dataset {
	knowhere::meta::TENSOR: - // query data
	knowhere::meta::ROWS: - // rows of queries
	knowhere::meta::DIM: - // dimension
	}
	
	Config {
	
	knowhere::meta::RADIUS: -
	
	knowhere::meta::RANGE_FILTER: -
	
	}

OUTPUT	Dataset {
	knowhere::meta::IDS: - // result IDs with length LIMS[nq]
	knowhere::meta::DISTANCE: - // result DISTANCES with length LIMS[nq]
	knowhere::meta::LIMS: - // result offset prefix sum with length nq + 1
	}

LIMS is with length "nq+1", it's the offset prefix sum for result IDS and result DISTANSE. The length of IDS and DISTANCE are the same but variable.

Suppose N queried vectors are with label: {0, 1, 2, ..., n-1}

The result counts for each queried vectors are: {r(0), r(1), r(2), ..., r(n-1)}

Then the data in LIMS will be like this: {0, r(0), r(0)+r(1), r(0)+r(1)+r(2), ..., r(0)+r(1)+r(2)+...+r(n-1)}

The total range search result num is: LIMS[nq]

The range search result for each query vector is: IDS[lims[n], lims[n+1]) and DISTANCE[lims[n], lims[n+1])

The memory used for IDS, DISTANCE and LIMS are allocated in Knowhere, they will be auto-freed when Dataset deconstruction.

3. BruteForce::RangeSearch()

This API does range search for no-index dataset, it returns all unsorted results with distance "better than radius" (for IP: > radius; for others: < radius).

PROTO	`static DatasetPtr`
	`RangeSearch(const DatasetPtr base_dataset,`
	`const DatasetPtr query_dataset,`
	`const Config& config,`
	`const faiss::BitsetView bitset);`

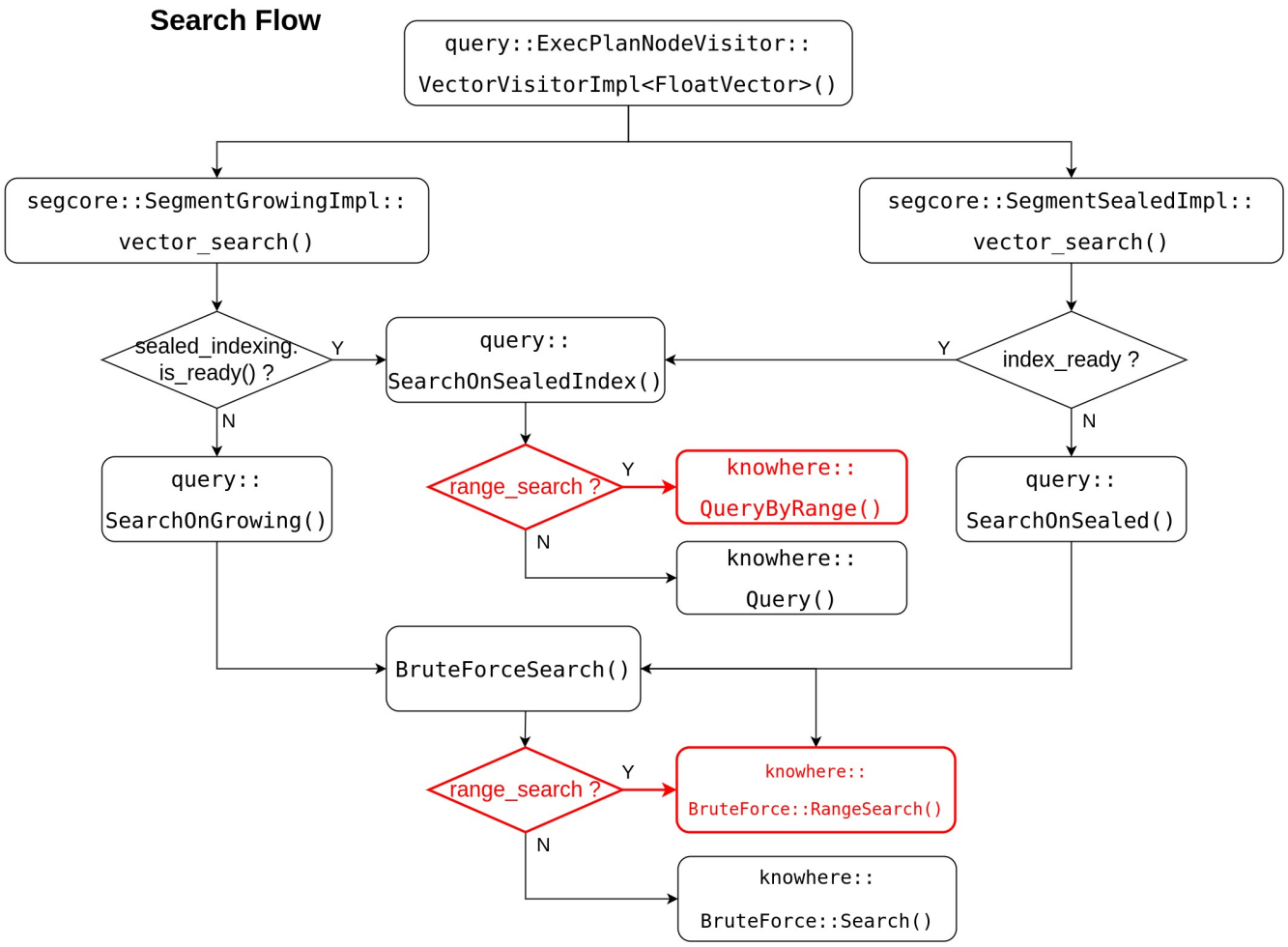
INPUT	Dataset {
	knowhere::meta::TENSOR: - // base data
	knowhere::meta::ROWS: - // rows of base data
	knowhere::meta::DIM: - // dimension
	}
	
	Dataset {
	knowhere::meta::TENSOR: - // query data
	knowhere::meta::ROWS: - // rows of queries
	knowhere::meta::DIM: - // dimension
	}
	
	Config {
	
	knowhere::meta::RADIUS: -
	
	knowhere::meta::RANGE_FILTER: -
	
	}

OUTPUT	Dataset {
	knowhere::meta::IDS: - // result IDs with length LIMS[nq]
	knowhere::meta::DISTANCE: - // result DISTANCES with length LIMS[nq]
	knowhere::meta::LIMS: - // result offset prefix sum with length nq + 1
	}

The output is as same as QueryByRange().

Segcore

Segcore search flow will be updated as this flow chart, range search related change is marked RED.



Segcore uses radius parameter's existence to decide whether to run search, or to run range search.

- when param "radius" is set, range search is called;
- otherwise, search is called.

For API query::SearchOnSealedIndex() and BruteForceSearch(), they do like following:

1. pass radius parameters (radius / range_filter) to knowhere
2. get all unsorted range search result from knowhere
3. for each NQ's results, do heap-sort
4. return result Dataset with TOPK results

Whatever do range search or search, the output structure are same:

- query::SearchOnSealedIndex() => SearchResult
- BruteForceSearch() => SubSearchResult

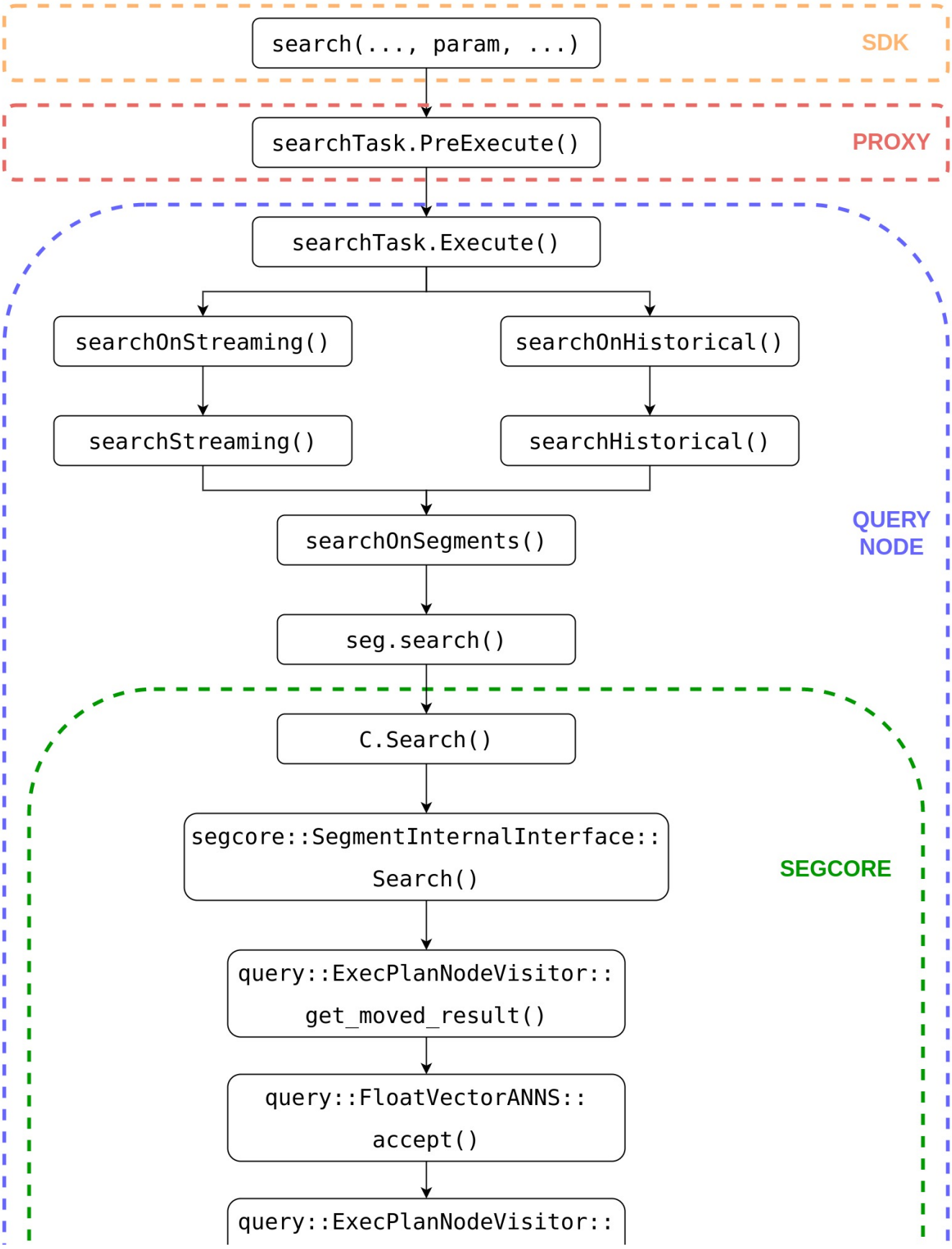
Both SearchResult and SubSearchResult contain TOPK sorted result for each NQ.

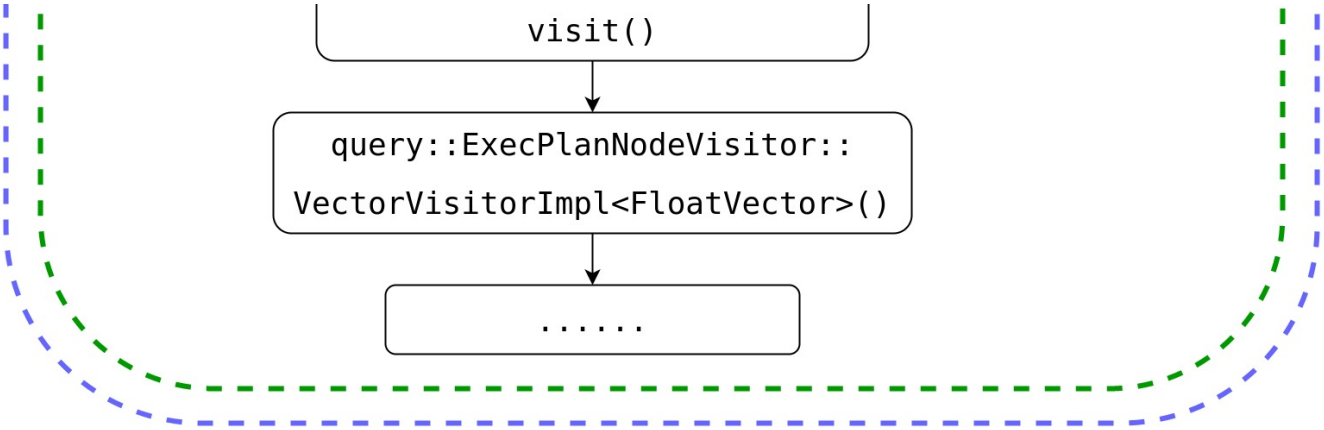
- seg_offsets_: with length "nq * topk", -1 is filled in when no enough result data
- distances_: with lengh "nq * topk", data is undefined when no enough result data

Milvus

Range search completely reuses the call stack from SDK to segcore.

Search Flow


search(..., param, ...)
SDK
searchTask.PreExecute()
PROXY
searchTask.Execute()
searchOnStreaming()
searchOnHistorical()
searchStreaming()
searchHistorical()
searchOnSegments()
QUERY NODE
seg.search()
C.Search()
segcore::SegmentInternalInterface::
Search()
SEGCORE
query::ExecPlanNodeVisitor::
get_moved_result()
query::FloatVectorANNS::
accept()
query::ExecPlanNodeVisitor::



How to get more than 16384 range search results ?

With this solution, user can get maximum 16384 range search results in one call.

If user wants to get more than 16384 range search results, they can call range search multiple times with different range_filter parameter (use L2 as an example)

- NQ = 1

1st call with (range_filter = 0.0, radius = inf), get result distances like this:

{d(0), d(1), d(2), ..., **d(n-1)**}

2nd call with (range_filter = **d(n-1)**, radius = inf), get result distances like this:

{d(n), d(n+1), d(n+2), ..., **d(2n-1)**}

3rd call with (range_filter = **d(2n-1)**, radius = inf), get result distances like this:

{d(2n), d(2n+1), d(2n+2), ..., **d(3n-1)**}

...

- NQ > 1 (suppose NQ = 2)

1st call with (range_filter = 0.0, radius = inf), get result distances like this:

{d(0,0), d(0,1), d(0,2), ..., **d(0,n-1)**, d(1,0), d(1,1), d(1,2), ..., **d(1,n-1)**}

2nd call with (range_filter = **min{d(0,n-1), d(1,n-1)}**, radius = inf), get result distances like this:

{d(0,n), d(0,n+1), d(0,n+2), ..., **d(0,2n-1)**, d(1,n), d(1,n+1), d(1,n+2), ..., **d(1,2n-1)**}

3rd call with (range_filter = **min{d(0,2n-1), d(1,2n-1)}**, radius = inf), get result distances like this:

{d(0,2n), d(0,2n+1), d(0,2n+2), ..., **d(0,3n-1)**, d(1,2n), d(1,2n+1), d(1,2n+2), ..., **d(1,3n-1)**}

...

The result of each iteration will have some duplication with the result of previous iteration, user need do duplication check and remove them.

Compatibility, Deprecation, and Migration Plan(optional)

This is a new functionality, there is no compatibility issue.

Test Plan(required)

Knowhere

1. Add new unittest
2. Add benchmark to test range search runtime and recall
3. Add benchmark to test range search QPS

There is no public dataset for range search. I have created range search data set based on sift1M and glove200.

You can find them in NAS:

1. test/milvus/ann_hdf5/binary/sift-4096-hamming-range.hdf5
    a. base dataset and query dataset are identical with sift1m
    b. radius = 291.0
    c. result length for each nq is different
    d. total result num 1,063,078
2. test/milvus/ann_hdf5/sift-128-euclidean-range.hdf5
    a. base dataset and query dataset are identical with sift1m
    b. radius = 186.0 * 186.0
    c. result length for each nq is different
    d. total result num 1,054,377
3. test/milvus/ann_hdf5/sift-128-euclidean-range-multi.hdf5
    a. base dataset and query dataset are identical with sift1m
    b. ground truth IDs and Distances are identical with sift1m
    c. each nq's radius is set to the last ground truth distance
    d. result length for each nq is 100
    e. total result num 1,000,000
4. test/milvus/ann_hdf5/glove-200-angular-range.hdf5
    a. base dataset and query dataset are identical with glove200
    b. radius = 0.52
    c. result length for each nq is different
    d. total result num 1,060,888

Segcore

1. Add new range search unittest

Milvus

1. use sift1M/glove200 dataset to test range search (radius = max_float / -1.0), we expect to get identical results as search

Rejected Alternatives(optional)

The previous proposal of this MEP is let range search return all results with distances better than a "radius".

The project implementation of the previous proposal is too complicated to achieve comparing with current proposal.

Adv.

1. Get all range search result in one call

Cons:

1. Need implement Merge operation for chunk, segment, query node and proxy
2. Memory explosion caused by Merge
3. Many API modification caused by invalid topk parameter

Others

Because the result length of range search from knowhere is variable, knowhere plan to afford another API to return the range search result count for each NQ.

If there is user request to get all range search result in one call, query node team will afford another solution to save range search output of knowhere to S3 directly.

References(optional)

1. Knowhere Range Search Algorithm Introduction
2. Range Search Introduction