MEP 34 -- IDMAP/BinaryIDMAP Enhancement

Current state: ["Declined"]

ISSUE: #17754

PRs: #240 #243 #18277

Keywords: IDMAP, BinaryIDMAP, brute force search, chunk

Released: Milvus-v2.2.0

Summary(required)

In this MEP, we put forward an IDMAP/BinaryIDMAP Enhancement proposal that let knowhere index type IDMAP/BinaryIDMAP to hold an external vector data pointer instead of adding real vector data in.

This Enhanced IDMAP/BinaryIDMAP can be used for growing segment searching to improve code reuse and reduce code maintenance effort.

Motivation(required)

Generally no one will create IDMAP/BinaryIDMAP index type for sealed segment, because it does not bring any search performance improvement but consumes identical size of memory and disk.

The only reasonable use scenario for IDMAP/BinaryIDMAP is for growing segment. But if create an IDMAP/BinaryIDMAP index in a normal way, it will consume lots of resources, because it will involve index node (to create index file), data node (to save index file to S3) and rootcoord / indexcoord / datacoord (to coordinate all these operations).

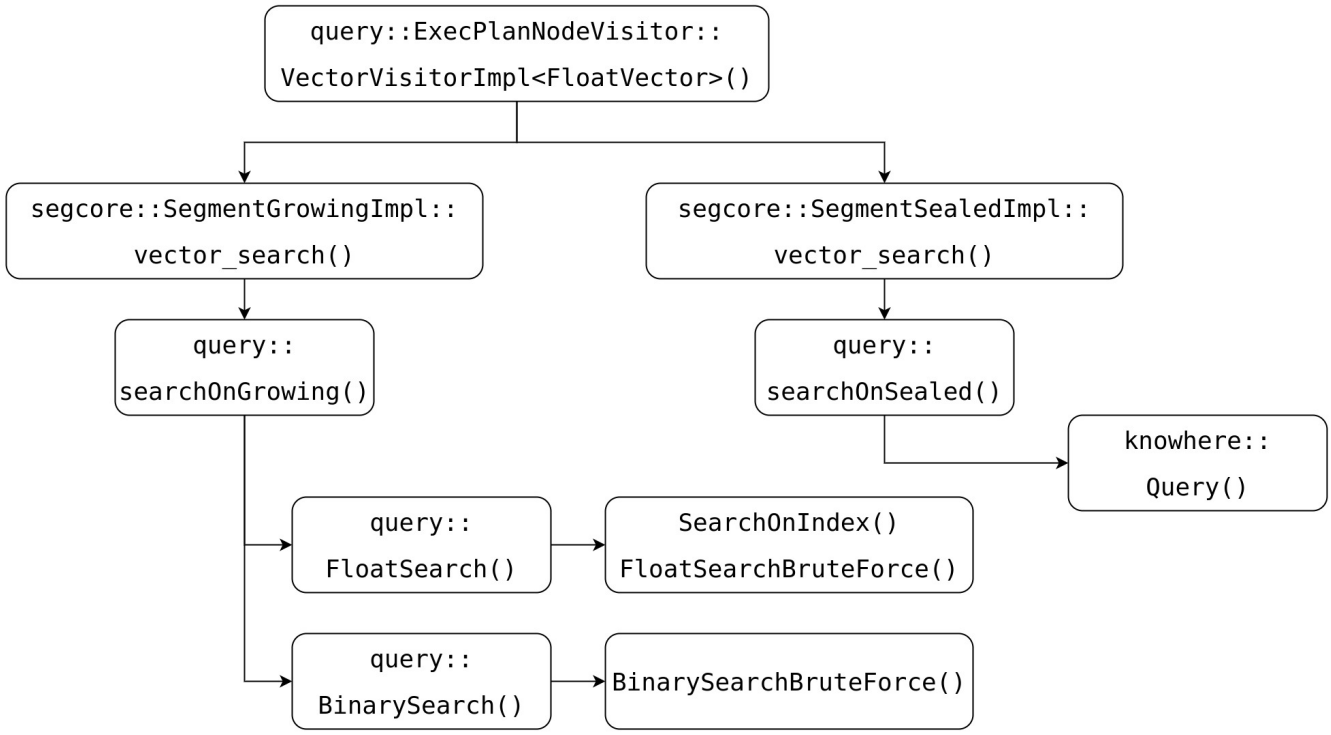
So Milvus uses following 2 strategies for growing segment searching:

- small batch index for fully-filled chunks (this functionality is disabled for some particular reason)
- brute force search for partial-filled chunks and no indexed fully-filled chunks (copied from knowhere IDMAP)

The advantage of this solution is resource saving, except query node, no other nodes will be involved in; while the shortcoming is code duplication.

See following "Search Flow" chart, `FloatSearchBruteForce` and `BinarySearchBruteForce` are copied from knowhere::IDMAP/BinaryIDMAP's interface Query() and modified a little. This will introduce more code maintenance effort. And when realize new feature on IDMAP/BinaryIDMAP in Knowhere, such as range search, we have to also copy these codes implementation to Milvus.

Search Flow



If we enhance IDMAP/BinaryIDMAP, not to add real vector data in, but only hold an external vector data pointer in the index, we can use knowhere::IDMAP /BinaryIDMAP's interface Query() directly without any costs. User need guarantee that the memory is contiguous and safe.

In this way:

- no CPU time, memory and disk consumption when creating index
- resource saving, only query node is involved in
- no code duplication for growing segment search
- unified search result for sealed segment and growing segment

Proposal 1 (only take IDMAP as an example)

Reuse index IDMAP, add new interface in both knowhere and faiss.

1. Faiss adds new field "codes_ex" and new interface "add_ex()" for structure IndexFlat. In IndexFlat, "codes" and "codes_ex" are mutual exclusive, user cannot set both of them.
2. Knowhere adds a new interface `AddExWithoutIds()` for IDMAP.
3. In Milvus, re-write API "FloatSearchBruteForce()", let it use enhanced IDMAP to search.

Advantages: Little code change

Cons: Need add new interfaces in both Faiss and Knowhere

Proposal 2 (only take IDMAP as an example)

Add new index type IDMAP_EX, add new interface in faiss.

1. Faiss adds new field "codes_ex" and new interface "add_ex()" for structure IndexFlat. In IndexFlat, "codes" and "codes_ex" are mutual exclusive, user cannot set both of them.
2. Knowhere adds a new index type IDMAP_EX, which use API "add_ex()" to insert vector data.
3. In Milvus, re-write API "FloatSearchBruteForce()", let it use new IDMAP_EX to search.

Advantages: No new interface in Knowhere

Cons: Need add new index type in Knowhere, most part of code of this index is duplicate with IDMAP

Proposal 3 (only take IDMAP as an example)

Add a new wrapper API in knowhere to call faiss brute force search for all metric types

Advantages: No code change in faiss, only one new interface in Knowhere

Cons: This change dis-obey knowhere's design concept. By now all operations in knowhere is for an index, but this API is for all metric types, not for an index.

Since both Proposal 1 and Proposal 2 need add new interface in Faiss, and Milvus team think this will break Faiss's encapsulation, we adopt Proposal 3.

Public Interfaces(optional)

Add new interface only in Knowhere

```
class BruteForce {
 public:
    static DatasetPtr
    Search(const DatasetPtr base_dataset,
           const DatasetPtr query_dataset,
           const Config& config,
           const faiss::BitsetView bitset);

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

BruteForce::Search() is wrapper API to call Faiss brute force search for all metric types.

BruteForce::RangeSearch() is wrapper API to call Faiss brute force range search for all metric types.

Design Details(required)

For growing segment in segcore, all brute force search related code can be removed, call knowhere's BruteForceSearch() and BruteForceRangeSearch() instead.

Compatibility, Deprecation, and Migration Plan(optional)

- This MEP will be transparent for users, and will not introduce any compatibility issue.

Test Plan(required)

New testcases are added in Knowhere to test "BruteForceSearch()" and "BruteForceRangeSearch()".

- in knowhere/unittest/test_bruteforce.cpp

No extra testcases need be added in Milvus because current growing segment search testcases can cover this change.

Search result and performance will be identical with before.

References(optional)

Briefly list all references