MEP 6 -- Support vector output fields design

Current state: Accepted

ISSUE: #6299

PRs: #6570 #6598 #6671 #7102

Keywords: Query / Search / Vector

Released: Milvus 2.0rc3

Summary

This project is to use minimal memory, let ***query*** support to return vector field in output.

Motivation

In Milvus 2.0rc1, ***query*** does not support return vector field in output. If ***query*** request's output fields contain float vector or binary vector, proxy will error out.

This is for the consideration of memory consumption, because vector field with big dimension will occupy hundreds of times of memory comparing with scalar

field. So generally ***load_collection*** or ***load_partition*** only load scalar fields' raw data into memory. Vector fields' raw data is loaded into memory only in 3 cases:

1. streaming segment
2. vector field's index type is FLAT
3. vector field's index has not been created

Only if vector's raw data has been loaded into memory, ***query*** can return vector field in output.

But ***query*** need this capability to return vector's raw data, for example tester can use this to check the correctness of inserted data.

Currently ***search*** also does not support return vector field in output, but we don't plan to enhance ***search*** in this project. If users need to get the vector data after

search returns ID, they can call **query** to get it.

If there is real requirement from users to let ***search*** return vector in output, we can achieve this in SDK level.

Design Details

Query supporting vector field in output can be divided into 2 steps:

1. in load segment stage, create VectorFieldInfo for vector fields, and save it into segment struct
2. in the end of query stage,
    a. load vector field's data if needed
    b. get vector data, fill in query result

- Add new field *VectorFieldInfo* into *segment* struct to return vector field related information

```
type VectorFieldInfo struct {
    mu              sync.RWMutex
    fieldID         UniqueID
    fieldBinlog     *datapb.FieldBinlog
    rowDataInMemory bool
    rawData         map[string]storage.FieldData  // map[binlogPath]FieldData
}

type Segment struct {
    ... ...
    vectorFieldInfos map[UniqueID]*VectorFieldInfo
}
```

- Add new interface in *segment_loader*

```
// load vector field's data from info.fieldBinlog, save the raw data into info.rawData
func (loader *segmentLoader) loadSegmentVectorFieldData(info *VectorFieldInfo) error {
```

- Add new interface in *query_collection*

```
// For vector output fields, load raw data from fieldBinlog if needed,
// get vector raw data via result.Offset from *VectorfieldInfo, then
// fill vector raw data into result
func (q *queryCollection) fillVectorFieldsData(segment *Segment, result *segcorepb.RetrieveResults) error
```

We also enhanced ***query*** to support wildcard in output fields.

- "*" - means all scalar fields
- "%" - means all vector fields

For example, A/B are scalar fields, C/D are vector fields, duplicated fields are automatically removed.

- output_fields=["*"] ==> [A,B]
- output_fields=["%"] ==> [C,D]
- output_fields=["*","%"] ==> [A,B,C,D]
- output_fields=["*",A] ==> [A,B]
- output_fields=["*",C] ==> [A,B,C]

Original vector data storage public interface and struct

Public Interfaces. It may be discussed and changed in future.

```
type ChunkManager interface {
        GetPath(key string) (string, error)
        Write(key string, content []byte) error
        Exist(key string) bool
        Read(key string) ([]byte, error)
        ReadAt(key string, p []byte, off int64) (n int, err error)
}
```

A VectorFileManager implements FileManager interface and add a method to download vector file from remote and deserialize its content, finally save pure vector to local storage.

```
type VectorChunkManager struct {
        localChunkManager  ChunkManager
        remoteChunkManager ChunkManager
}

func NewVectorChunkManager(localChunkManager ChunkManager, remoteChunkManager ChunkManager) *VectorChunkManager
```

localChunkManager is responsible to local file manager. And can be implements with golang os library. The path of local chunk manager is config in **milvus.yaml** with key **storage.path**.
remoteChunkManager is responsible for cloud storage or remote server storage, and will be implemented with minio client now.

When the offset of vector is obtained, we can get origin vector data from local vector data file.

Get the vector the ID through the following process

1.Get segment's id size in each binlog and vector file names when load_segment. The binlogs file will be sorted by file name's last id to guarantee the order is increasing. Suppose we get sizes are 300, 300, 400, 500.

2.Get the id offset in segment in C layer. Suppose we get an offset 700.

3.We can know the vector we want to get is in 3rd vector files. for 300+300 <700<300+300+400

4.Get the 3rd file in to memory and deserialize out pure vector. Save the vector to local storage. Release the memory usage.

5.Mmap the file to memory, and get the data of offset 100. The data length differs data type and dim.

Test Plan

Do ***query*** / ***search*** (with vector field in output fields) in all kinds of combinations of following scenarios, check the correctness of result.

1. float vector or binary vector
2. with/wo index
3. all kinds of index type