team11 final project report

109062127丁旭寬 109062319楊智明 109062301葉宥忻

Implement

1. IVF_SQ8_DIRECT
   系統架構：參考PASE的IVF_FLAT以及VanillaDB既有的Index寫法（例如HashIndex、IndexSelectPlan、ConstantRange、SearchKey、executeTrainIndex等等），不一樣的地方例如我們不用meta page（而是只從vanilladb.properties取得）、我們的centroid page使用spanned record、我們的data page包含原table所有的field（原來只包含key跟record id；這樣在搜尋資料的時候比較連續，可以降低I/O的次數）。我們另外實作了InsertionSortPlan，在知道KNN的K很小的情形下，不需要在disk上跑merge sort才能知道前K名是誰。
   SQ8_DIRECT: 我們發現sift.txt裡面都是介於0~218的整數，所以應該可以做適當的quantization。我們一度以為直接用8個bit來存是作弊，但我們後來發現Faiss根本也有類似的東西：

```
enum QuantizerType {
    QT_8bit,         ///< 8 bits per component
    QT_4bit,         ///< 4 bits per component
    QT_8bit_uniform, ///< same, shared range for all dimensions
    QT_4bit_uniform,
    QT_fp16,
    QT_8bit_direct, ///< fast indexing of uint8s
    QT_6bit,        ///< 6 bits per component
    QT_bf16,
};
```

   換句話說，我們可以在完全不犧牲recall的情形下，進一步降低I/O的次數。所以，我們新增了ByteVector相關的type跟method，來符合我們的要求。我們只有quantize存在data page的vector，centroid vector一樣是用float來存。
2. Kmeans: 首先使用來RandomNonRepeatGenerator來從0~899999選不重複的數字

```
        RandomNonRepeatGenerator RNRG = new RandomNonRepeatGenerator(SiftBenchConstants.NUM_ITEMS);
        Map<Integer, Integer> M = new HashMap<>();
        for (int i = 0; i < IVFFlatIndex.NUM_CENTROIDS; ++i){
            int random_number = RNRG.next();
            M.put(random_number,i);
        }
```

   接著掃過table，如果遇到剛剛選到的點，就把他設為一個cluster的中心

```java
        Plan test_tp = new TablePlan(tableName, tx);
        Scan test_ts = test_tp.open();
        test_ts.beforeFirst();
        while(test_ts.next()){
            int index = (Integer)test_ts.getVal(fldName:"i_id").asJavaVal();
            if(M.containsKey(index)){
                VectorConstant v = new VectorConstant((float[])test_ts.getVal(fldName:"i_emb").asJavaVal());
                idx.setCentroidVector(M.get(index), v);
            }
        }
        test_ts.close();
```

接著refine剛剛選到的中心iteration次，每次refine都會掃過整個table一次

```java
        // TODO: refine the centroids by K-means. It is recommended to log each iteration
        int iteration = 2;

        for (int i = 0; i < iteration; i++){
            System.err.print("Iteration " + i + "\n");
            Plan tp = new TablePlan(tableName, tx);
            Scan ts = tp.open();
```

3. SIMD
   我們使用了sub、fma來計算Euclidean distance，其中針對除以SPECIES.length()的餘數部分再計算一次差平方做加總，最後開根號並回傳

```java
    @Override
    protected double calculateDistance(VectorConstant vec) {
        int i = 0;
        FloatVector sum = FloatVector.zero(SPECIES);
         for (; i < SPECIES.loopBound(vec.dimension()); i += SPECIES.length()) {
            FloatVector v = FloatVector.fromArray(SPECIES, vec.asJavaVal(), i);
            FloatVector q = FloatVector.fromArray(SPECIES, query.asJavaVal(), i);
            FloatVector diff = v.sub(q);
            sum = diff.fma(diff, sum);
        }
        double sum_d = sum.reduceLanes(VectorOperators.ADD);

        for (i=0 ; i < vec.dimension(); i++) {
           double diff = query.get(i) - vec.get(i);
           sum_d += diff * diff;
        }

        return Math.sqrt(sum_d);
    }
```

Other improvement

1. 我們發現public VectorConstant(byte[] bytes)是CPU效能瓶頸，如下圖。我們一度想把這個constructor也SIMD化，但後來發現用java.nio優化比較好。

```
Top methods over 60013 ms (0 ms paused), with 26653 counts:
Rank    Self    Stack   Method
1       21%     21%     org.vanilladb.core.sql.VectorConstant.<init>
2       11%     11%     org.vanilladb.core.storage.file.io.javanio.JavaNioFileChannel.write
3       9%      9%      org.vanilladb.core.storage.file.io.javanio.JavaNioFileChannel.append
4       6%      6%      org.vanilladb.core.storage.file.io.javanio.JavaNioFileChannel.<init>
5       5%      19%     org.vanilladb.core.storage.buffer.BufferMgr.pin
6       3%      3%      org.vanilladb.core.sql.VectorConstant.asBytes
7       3%      4%      org.vanilladb.core.storage.tx.concurrency.LockTable.sLock
8       2%      27%     org.vanilladb.core.storage.file.Page.getVal
9       2%      4%      org.vanilladb.core.storage.tx.concurrency.LockTable.isLock
10      2%      24%     org.vanilladb.core.sql.Constant.newInstance
11      2%      2%      org.vanilladb.core.storage.tx.concurrency.LockTable.releaseLock
12      2%      2%      org.vanilladb.core.storage.file.io.javanio.JavaNioFileChannel.read
13      1%      1%      org.vanilladb.core.storage.tx.concurrency.LockTable.getAnchor
14      1%      1%      org.vanilladb.core.storage.file.io.javanio.JavaNioFileChannel.close
15      1%      3%      org.vanilladb.core.storage.file.FileMgr.delete
16      1%      1%      org.vanilladb.core.storage.record.RecordPage.close
16      1%      19%     org.vanilladb.core.storage.record.RecordPage.<init>
17      1%      1%      org.vanilladb.core.storage.tx.concurrency.LockTable.prepareLockers
```

2. 我們一度想要讓尋找最接近的centroid這個操作在多個thread上跑，因為它是”CPU”效能瓶頸之一。結果因為實作寫得不好，不小心pin了太多buffer，只好作罷。
3. 在原版Kmean做refine的時候，我們一開始是掃過整個table才更新一次centroid，但這樣更新一次要快將近20分鐘，centriod更新效率不高。因此，我們做了簡單的優化，做法是每掃10000個record就更新一次centroid，這樣只需掃一次table就能更新90次centroid。

Experiments

Environment:

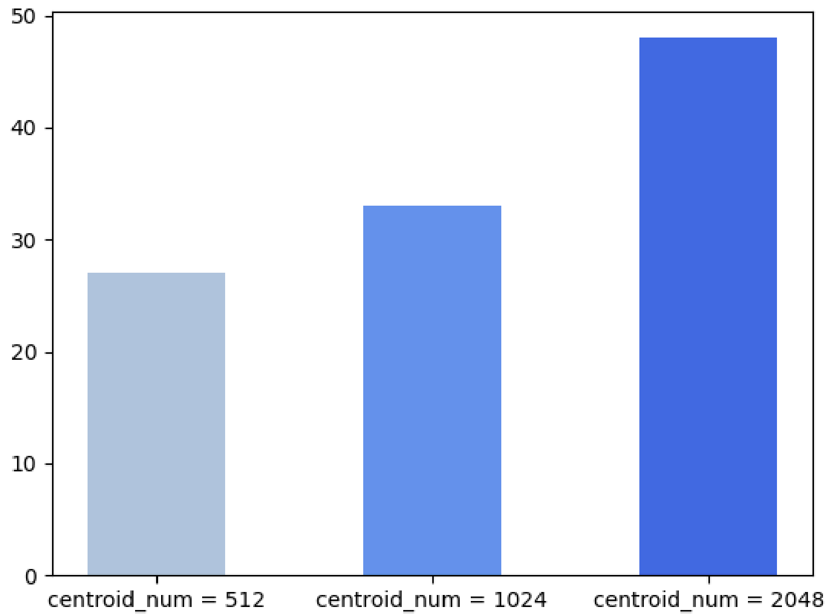
Intel Core i5-6400 CPU @ 2.7GHz, 16 GB RAM, 224 GB SSD, Windows 11

Different centroid_num (probe bucket = 8)

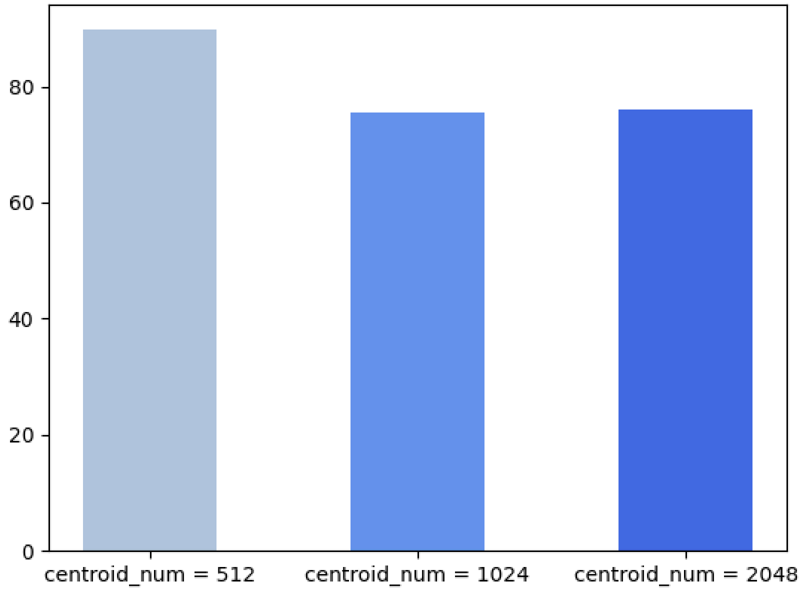
Results:

cen_num	512	1024	2048
commit	27	33	48
recall	89.77%	75.46%	75.95%

Graph(Committed):



Graph(Recall):

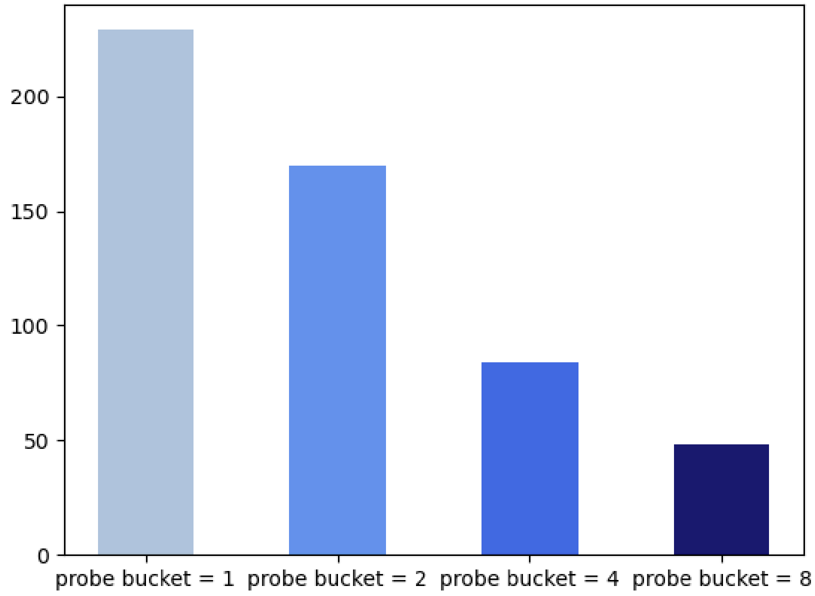


Different probe bucket (centroid_num = 2048)

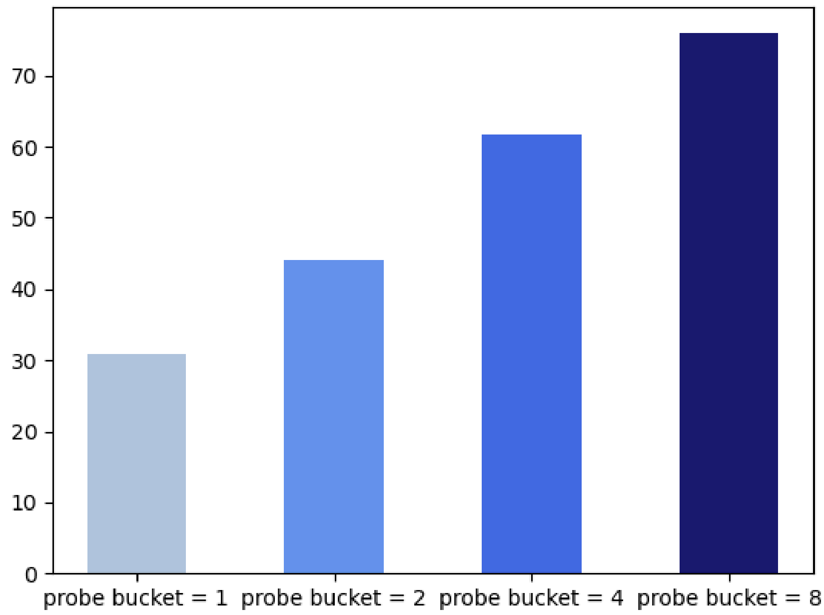
Result:

pro_bucket	1	2	4	8
commit	229	170	84	48
recall	30.94%	44.10%	61.72%	75.95%

Graph(Committed):

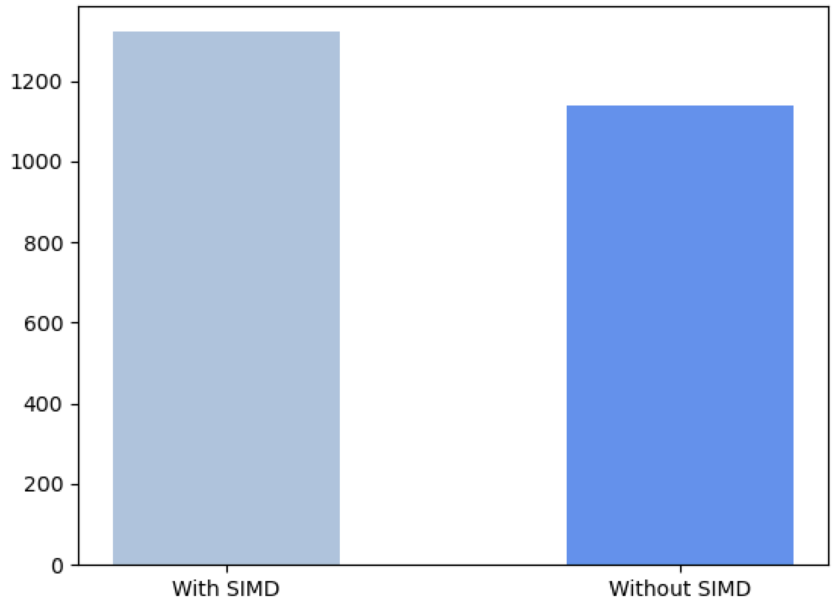


Graph(Recall):



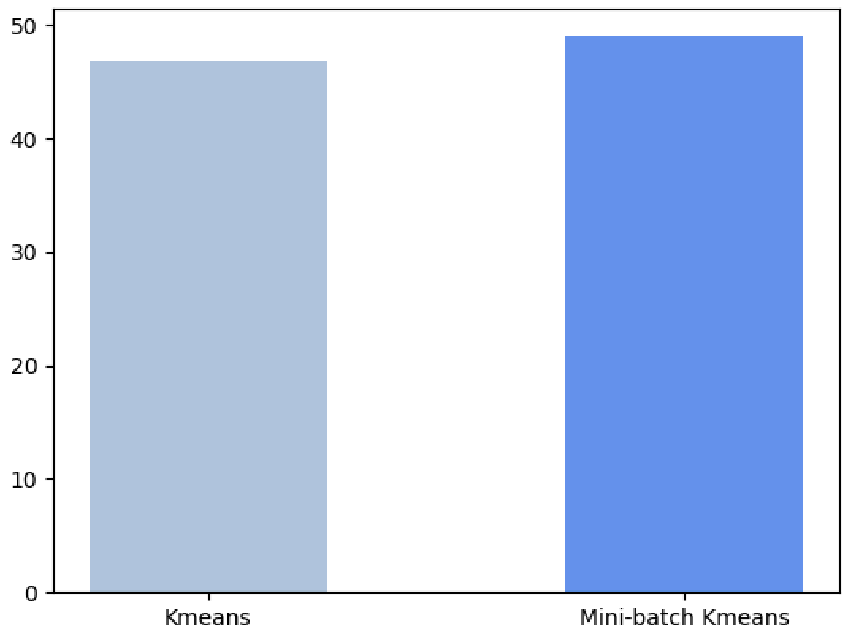
With/Without SIMD (centroid_num = 512, probe bucket = 8)

Graph(Committed):



Kmeans/Mini-Batch Kmeans (centroid_num = 512, probe bucket = 8)

Graph(Recall):



Conclusion

1. For centroid_num

    a. centroid_num越低，recall就越高。因為當cluster切的越大塊，KNN搜索的範圍就會越大，找到實際最近的機率就會越高
    b. centroid_num越低，commited就越低。因為搜索範圍越大塊，要接觸到的vector就會越多
2. For probe bucket
    a. probe bucket越高，recall就越高。因為找的cluster越多，表示搜索範圍越大
    b. probe bucket越高，commited就越低。因為搜索範圍越大塊，要接觸到的vector就會越多
3. 由實驗結果可以看出有做SIMD能有效增加commit的txn
4. Mini-Batch Kmeans之recall比Kmeans高3%左右，這是因為Mini-Batch較頻繁更新centroid，故其收斂較快