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

```
74          RandomNonRepeatGenerator RNRG = new RandomNonRepeatGenerator(SiftBenchConstants.NUM_ITEMS);
75          Map<Integer, Integer> M = new HashMap<>();
76          for (int i = 0; i < IVFFlatIndex.NUM_CENTROIDS; ++i){
77              int random_number = RNRG.next();
78              M.put(random_number,i);
79          }
```

   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

```
81          Plan test_tp = new TablePlan(tableName, tx);
82          Scan test_ts = test_tp.open();
83          test_ts.beforeFirst();
84          while(test_ts.next()){
85              int index = (Integer)test_ts.getVal(fldName:"i_id").asJavaVal();
86              if(M.containsKey(index)){
87                  VectorConstant v = new VectorConstant((float[])test_ts.getVal(fldName:"i_emb").asJavaVal());
88                  idx.setCentroidVector(M.get(index), v);
89              }
90          }
91          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

```
93          // TODO: refine the centroids by K-means. It is recommended to log each iteration
94          int iteration = 2;
95
96          for (int i = 0; i < iteration; i++){
97              System.err.print("Iteration " + i + "\n");
98              Plan tp = new TablePlan(tableName, tx);
99              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

```
@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。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。
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。結果後來發現buffer不夠用，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

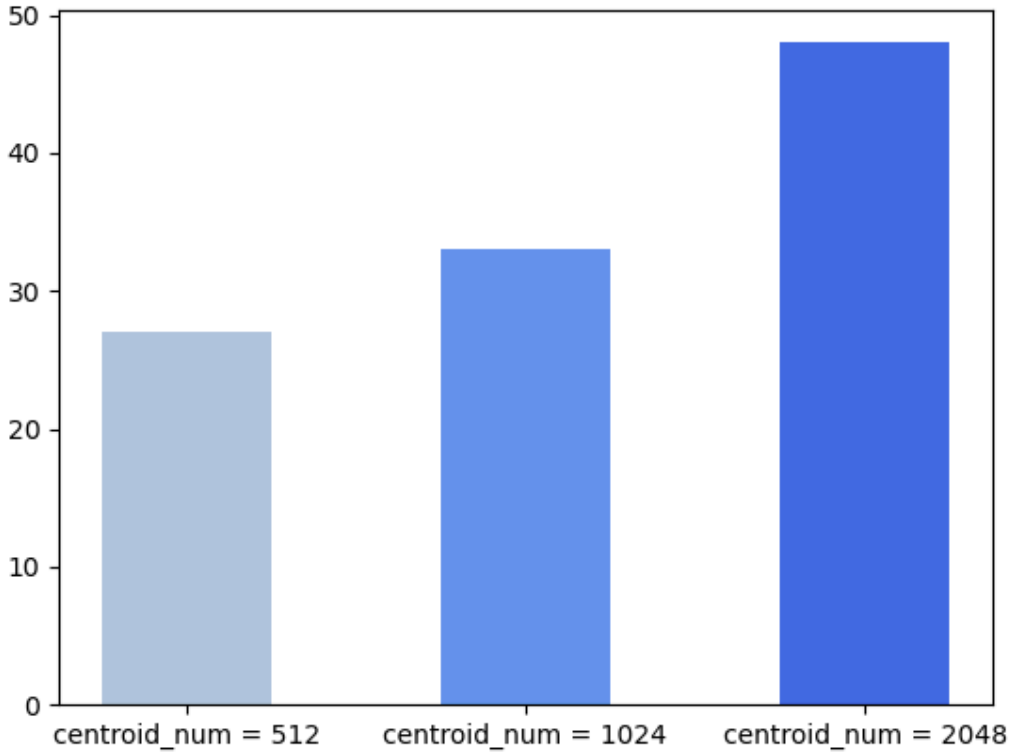
Environment:

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

Different centroid_num (probe bucket = 8)

(縱軸為一秒的commited數量)

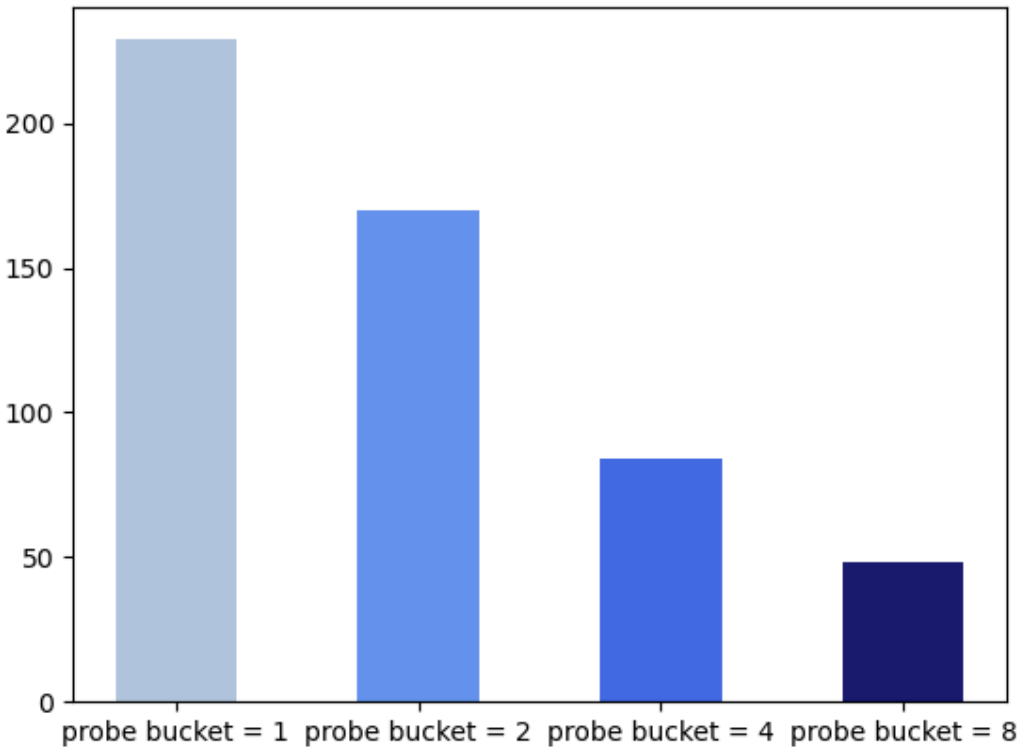
cen_num	512	1024	2048
commit	27	33	48
recall	89.77%	75.46%	75.95%



Different probe bucket (centroid_num = 2048)

(縱軸為一秒的commited數量)

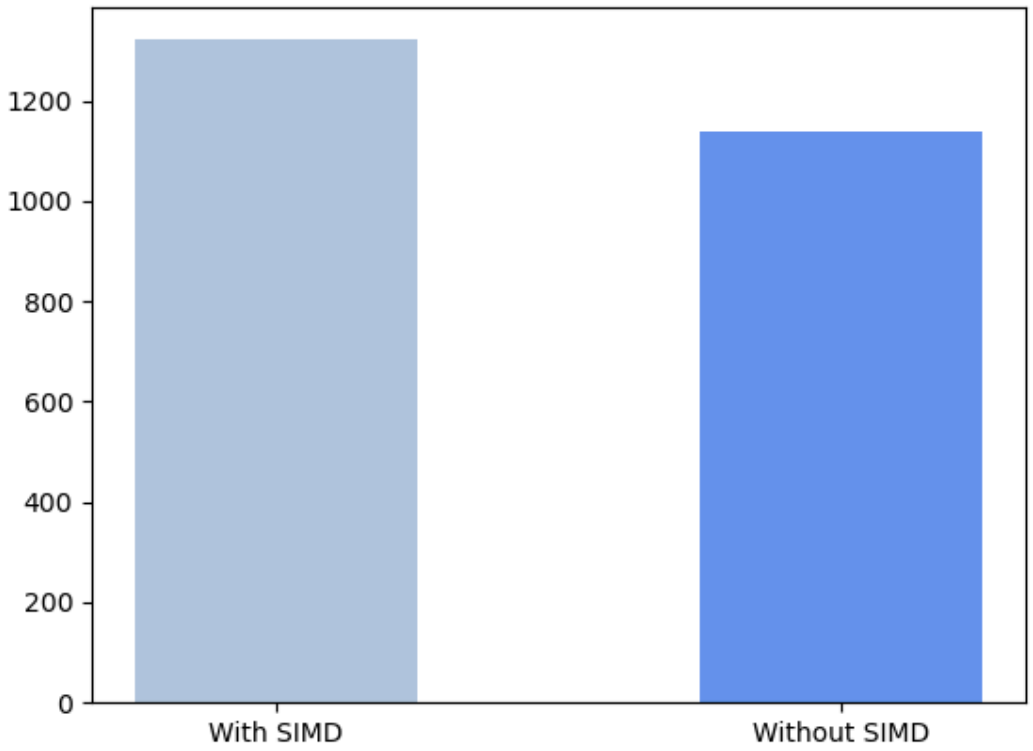
pro_bucket	1	2	4	8
commit	229	170	84	48
recall	null	null	61.72%	75.95%



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

with:commit = 1322

without: commit = 1138



Conclusion

1. 由實驗結果可以看出有做SIMD能有效增加commit的txn
2. 可看出centroid_num調越高，commit的txn也會越高，但recall會有些微的下降
3. 可看出probe bucket調越高，commit的txn會越低，但recall有上升的趨勢