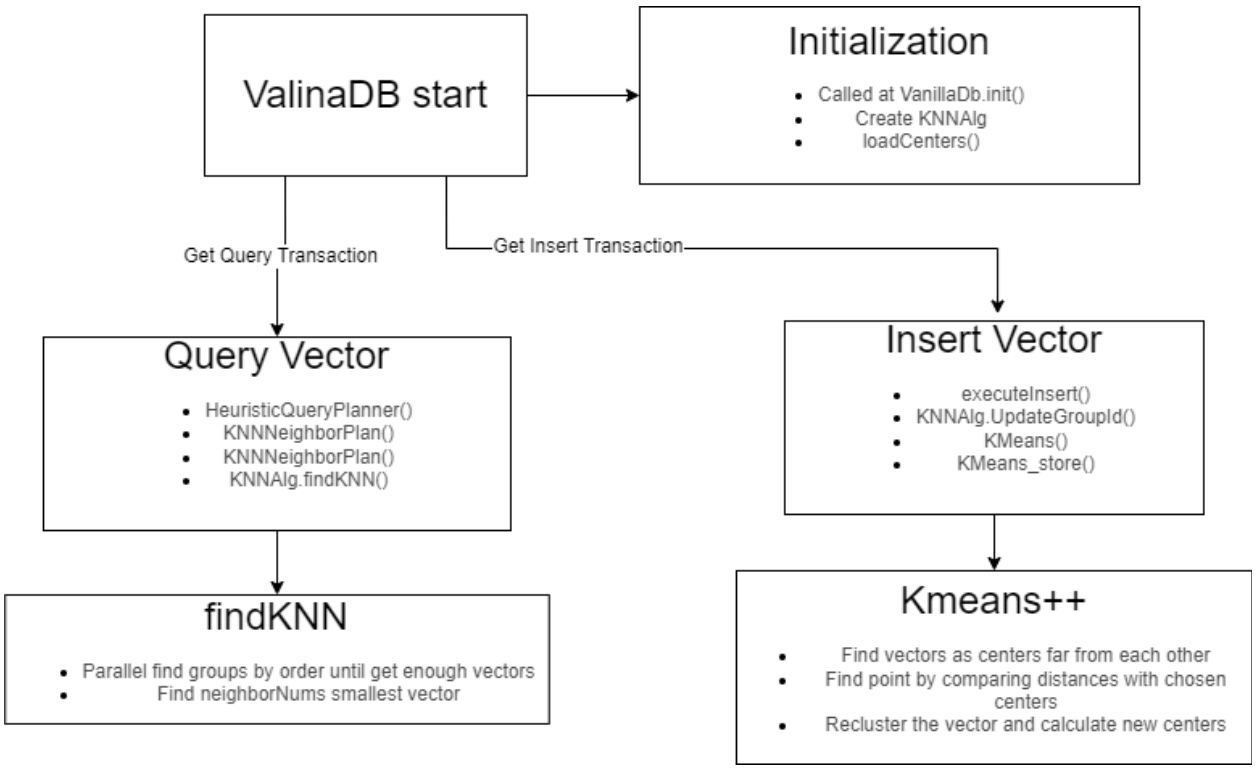
Team 16 Final Report

109062123 曹瀚文
109062323 吳聲宏
109062330 黃鈺臻

架構:



在這份project裡，為了實現加速搜尋vector，實作KMeans。主要分為三個部分。
首先在insert的部分，當table輸入了一定的量之後，KMeans()會被呼叫，尋找K個center。
在KMeans()的初始尋找時，原本我們會用KMeans++去做雖機挑選，讓centers可以差更遠，可以更快找到正確位置，但是後來試驗後發現，不去用KMeans++反而可以增加運算速度。
當query的時候，會找出離query vector最近的center，並從這個group裡面去尋找，最近的N個data，假如在最近的group找不滿N個，會從第二近的group尋找，一直下去。
在initialization的部分，會把代表centers的vectors從table 放進memory，以便之後query時的尋找速度。

SIMD:

```java
protected double calculateDistance2(VectorConstant vec) {
    double[] vec_d = vec.asJavaVal_d();

    int i = 0;
    DoubleVector sum = DoubleVector.zero(SPECIES);
    for (; i < SPECIES.loopBound(vec.dimension()); i += SPECIES.length()) {
        DoubleVector v = DoubleVector.fromArray(SPECIES, vec_d, i);
        DoubleVector q = DoubleVector.fromArray(SPECIES, query_d, i);
        DoubleVector diff = v.sub(q);
        sum = diff.fma(diff, sum);
    }
    double sum_d = sum.reduceLanes(VectorOperators.ADD);


    // Dealing with tail of dim % SPECIES.length()
    sum_d = 0;
    for (i=0 ; i < vec.dimension(); i++) {
        double diff = query.get(i) - vec.get(i);
        sum_d += diff * diff;
    }

    return sum_d;
}
```

KNNHelper:
在這個class裡面，主要是為了方便之後的實作。

- init()

  在init的部分會先去確認此function是否已經被呼叫過。沒有的話，會新增兩個table，一個是儲存centroid的table，包含group id，group size，和vector。vector是用來儲存center vector，而id和size是為了query可以方便尋找。另一個table是index table，紀錄record是屬於哪個group，或是哪個block。有被init過了就不用在做任何事
- updateGroupId()，updateGroupCenter()

  用來新增進index table或是center table。利用executeInsert()。
- queryGroupCenters()，queryRecord()

  queryGroupCenters()會列出所有的centers，從這之中可以選出需要找尋的groupId，用queryRecord()把屬於的record列出。

KNNAlg:

- KNNAlg()

  會去呼叫KNNHelper的init(), 和取得centers的資料
- UpdateGroupId()

  每次使用IndexUpdatePlanner的executeInsert會呼叫一次, 足夠多次的時候會呼叫KMeans(), 計算centers的位置。
- KMeans()

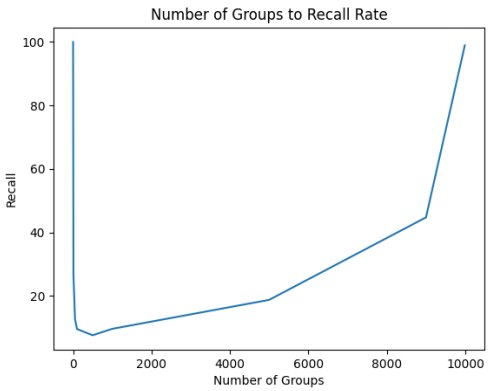
  先找出隨機的centers, 並找出所有data個別屬於哪個group。將新的center設為一個group裡面的平均值。一直重複這步驟。最後利用KNNHelper將算好的centers儲存回table。
- KMeans++()

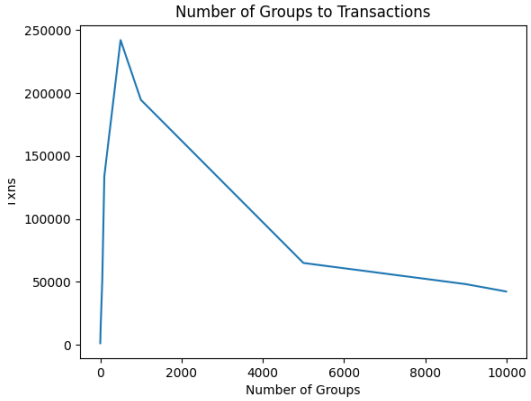
  KMeans()的進化版, 從隨機選取vector當起始點, 變成用距離的遠近增加被選中的機率。改成KMeans++()之後可以減少iteration的次數。但做實驗發現, 對uniform的資料使用KMeans++()好像跟單純KMeans()差不多。
- findKNN()

  從centers裡找到最接近query vector, 接著從找到的group裡尋找最接近的N個data, 用list回傳到我們新增的plan。我們之後有增加一個叫Multiplier的參數, 先會找Multiplier * N數, 再從這裏面找最近的vectors。這樣理論上可以增加recall, 但也會變慢。

實驗:

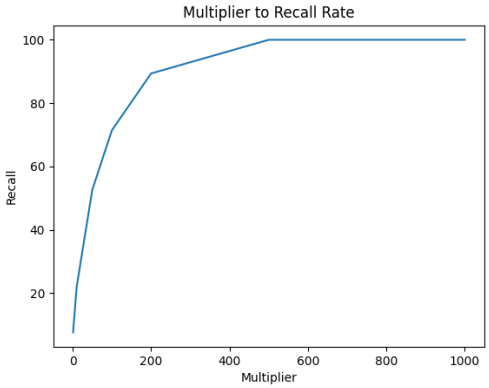
- NumGroups對Recall和transactions的影響 (multiplier = 1)

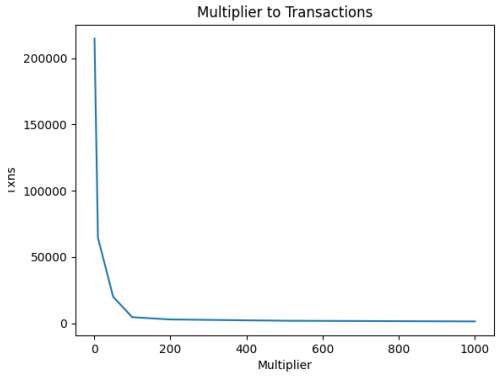




我們跑了實驗測試number of groups對正確率和速度的影響, 具體數值如上圖。我們發現Recall Rate和Transactions會承負相關。圖中的最低點(最高點)的數值為NumGroups = 500, 過500後會Recall會慢慢上升, Transactions會慢慢下降。

- Multiplier對Recall和transactions的影響 (numGroups = 200)





我們也對multiplier改數值並做測試，並歸納出雖然multiplier上升會增加recall rate，但Transaction數量也會一樣跟著降低。
看完圖表和比較，我們找出來決定NumGroup = 200, multiplier = 13

謝謝助教，助教辛苦了~