UM6P UM6P

Abstract

300 word description of the project

Paper Title

Karima Echihabi

November 17, 2022

PVLDB Reference Format:
. , 15(10): XXX-XXX, 2022.
https://doi.org/XX.XX/XXX.XXdoi:XX.XX/XXX.XX

1 Proposed Approach

We sincerely thank X, Y and Z.

References

[1] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB*, 12(2), 2018.

[2] PhDComics. Graduate Student Work Output. `https://phdcomics.com/comics/archive.php?comicid=124`, 2022.

Proceedings of the VLDB Endowment, Vol. 15, No. 10 ISSN 2150-8097.
https://doi.org/XX.XX/XXX.XXdoi:XX.XX/XXX.XX

Algorithm 1: KASHIF: STARTTHREADPOOL

Input: $JobArray = \{q_1, ..., q_n\}$, k, r_{th}, m the number of threads, $JobCounter$, $AllWorkersDone$.
Output: $KnnResults$.

Thread Array $Workers[m]$;
Boolean Array $WorkersState[m] \leftarrow \{False, ..., False\}$;
Integer $NumWorkingThreads$ = m;
Barrier Array $Barrier[m]$;
Boolean $WorkerDone \leftarrow True$;
Shared Queue
$KnnResults[n][k] \leftarrow \{\{+\infty_1, ..., +\infty_k\}_{q_1}, ..., \{+\infty_1, ..., +\infty_k\}_{q_n}\}$;
for $i \leftarrow 0$ **to** $m - 1$ **do**
    $Workers[i]$.INITWORKERTHREAD($JobArray, k$, $JobCounter, Barrier[i], KnnResults, WorkersState[i], NumWorkingThreads$);
// get incremental results
while *True* **do**
    for $i \leftarrow 0$ **to** $m - 1$ **do**
        if *!AtomicCompare*$(WorkersState[i], WorkerDone)$ **then**
            Coordinator waits on $Barrier[i]$;
    if *!AtomicCompare*$(NumWorkingThreads, 0)$ **then**
        // no more jobs and all workers finished their jobs.
        break;

Algorithm 2: HEURISTICKNNSEARCH

Input: A query vector q, k, N_{root}.
Output: k Nearest Vectors to q.

Queue $Knn[k] \leftarrow \{(null, \infty_1), ..., (null, \infty_k)\}$;
Node $N \leftarrow N_{root}$;
while *!*$N.IsLeaf()$ **do**
    $SP = N.SplitPolicy()$;
    $N = N.RouteToChildNode(q, SP)$;
$(v', bsf) = Knn[k]$;
foreach $v \in N.Vcetors()$ **do**
    if $D(v, q) \leq bsf$ **then**
        $Knn.SortedInsert(v, D(v, q))$;
    $(v', bsf) = Knn[k]$;
return *Knn;*

Algorithm 3: KASHIF: INITWORKERTHREAD

Input: $JobArray$, k, $JobCounter$, $Barrier$, $KnnResults$, $WorkerState$, $NumWorkingThreads$.

```
1 while True do
2     JobIdx = SyncFetchAndAdd(JobCounter);
3     if JobIdx > JobArray.Size() then
4         SyncFetchAndSub(NumWorkingThreads);
5         break; // no more jobs to do
6     q = JobArray[JobIdx];
7     ExactKnnSearch(q, k, KnnResults[JobIdx], Barrier, WorkerState);
```

Algorithm 4: ExactKnnSearch

Input: A query vectors q, k, knnResults, Barrier, N_{root}.

```
// local knn results, ordered by distance
```
Queue $Knn[k] \leftarrow \{(null, +\infty_1), ..., (null, +\infty_k)\}$;
Queue $pq \leftarrow \{\}$; `// priority queue`
```
/* last NN returned by incremental search is at position
   LastIncrResult - 1                                    */
```
Integer $LastIncrResult \leftarrow 0$;
Integer $UpdateStart \leftarrow 0$; `// where new results start`
Integer $UpdateEnd \leftarrow 0$; `// where new results end`
Boolean $NewIncrement \leftarrow False$;

$AtomicSet(WorkerState, False)$;
```
// thread is working
/* perform heuristic search and update knn results in global
   array                                                 */
```
$KnnResults \leftarrow$HeuristicKnnSearch(q, k);
```
// initialize priority queues
```
$pq.Add(N_{root})$;

while *!pq.Empty()* **do**
    NewIncrement = False;
    while *!NewIncrement and !pq.Empty()* **do**
        $N \leftarrow pq.Pop()$;
        for $pos \leftarrow LastIncrResult$ **to** $k - 1$ **do**
            $(v', bsf) \leftarrow Knn[pos]$;
            if $D_{lb}(N, q) > bsf$ **then**
                $LastIncrResult \leftarrow pos + 1$;
                $NewIncrement \leftarrow True$;
        if *NewIncrement* **then**
            $UpdateStart \leftarrow UpdateEnd$;
            $UpdateEnd \leftarrow LastIncrResult$;
        if $N.IsLeaf()$ **then**
            $(v', bsf) \leftarrow Knn[k]$;
            foreach $v \in N.Vcetors()$ **do**
                if $D(v, q) < bsf$ **then**
                    $Knn.SortedInsert(v, D(v, q))$;
        else
            foreach N' *in* $N.ChildNodes()$ **do**
                if $D_{lb}(N', q) < bsf$ **then**
                    $pq.Add(N')$;
    `// copy new results to global kNN Queue`
    $CopyResults(Knn, KnnReasults, UpdateStart, UpdateEnd)$;
    `// Wait for other threads to finish current increment`
    Thread blocks on *Barrier*; $AtomicSet(WorkerState, True)$;
    `// thread stoped working`