Parallel Construction of an N-tree

Ralf Hartmut Güting, August 17, 2023

Run this script with *SecondoTTYBDB* and prefix `@%` or `@&`.

This script builds an N-tree in parallel by constructing the trees of level 2 in a distributed manner while building the root node and the level 1 nodes directly in the script.

1 The Database

Database newyork12 on moma.

Preparations

```
let_ TripsCM200R = 'TripsCM200' ffeed5 filter[no_components(.Trip) > 0]
  consume;

restore Workers from Workers32

let myPort = 61239
```

```
let_ TripsCM200D = TripsCM200R feed addid
  ddistribute3["TripsCM200D", 128, TRUE, Workers]
```

We have 32 workers and wish to use a reasonable number of slots. So we use an N-tree of degree 25 and construct the two top levels with 25 * 25 = 625 centers.

2 Select Centers for the Root Node

```
let_ Centers25 = TripsCM200D dmap["", . feed some[1]] dsummarize some[25]
  addcounter[N1, 0] consume

query share("Centers25", TRUE, Workers)
```

3 Distribution to Root Centers (Level 0)

```
query TripsCM200D dcommand['query memclear()'] filter[.Ok] count;
query TripsCM200D dcommand['query meminit(1800)'] filter[.Ok] count;

query TripsCM200D dlet["Centers25m", 'Centers25 feed mconsume']
  filter[.Ok] count
```

We build N-trees with a single leaf.

```
query TripsCM200D dlet["Centers25m_ntree", 'Centers25m
  mcreatentree7[Trip, 350, 1000]'] filter[.Ok] count
```

If we assign elements (Trips) just to their closest center, we get 25 partitions. However, we want to employ 80 workers. For better load balancing, each worker should process several slots. Therefore, we assign each element not only to the closest center, but also to one of 20 subgroups of this center, derived from slot numbers.

```
let_ Trips2501 = TripsCM200D dmap["",
  fun(slot: ARRAYFUNARG1, slotnumber: int)
  slot feed
  loopjoin[Centers25m_ntree Centers25m mclosestCenterN[.Trip]
    projectextend[N1; N1Dist: .CenterDistance]]
  extend[N1x: (.N1 * 20)  + (slotnumber mod 20)]
  consume]

let_ Trips2501b = Trips2501 partition["", .N1x, 500]
```

4 Select Centers of Level 1 Below the Root (Root Is Level 0)

Again, we create 20 instances of each set of centers at level 1, to match the partitioning of *Trips2501b*.

```
let_ Centers2501a =  Trips2501
  dmap["Centers2501", . feed sortby[N1] nest[N1; Tuples]
    extend[CenterCands: .Tuples afeed some[1] aconsume]
    remove[Tuples] unnest[CenterCands]]
  dsummarize
  sortby[N1]
  nest[N1; Tuples2]
  extend[Centers2: .Tuples2 afeed some[25] addcounter[N2, 0] aconsume]
  remove[Tuples2]
  intstream(0, 19) transformstream product
  extend[N1y: (.N1 * 20) + .Elem]
  unnest[Centers2]
  project[Trip_id, PickupTime, DropoffTime, TotalDistance, TotalDuration,
    SpeedKmh, Trip, CM, TID, N1, N1Dist, N1x, N2, N1y]
  consume

let_ Centers2501 = Centers2501a feed ddistribute2["", N1y, 500, Workers]
  partition["", .N1y, 0]
```

5 Distribution to Level 1 Centers

```
let_ Trips2502a = Trips2501b Centers2501
  areduce2["", fun(part1: AREDUCEARG1, part2: AREDUCEARG2)
    part2 funseq3[
      $1 feed mconsumeflob,
      $2 mcreatentree7[Trip, 350, 1000],
      part1 feed loopjoin[$3 $2 mclosestCenterN[.Trip]
        projectextend[N2; N2Dist: .CenterDistance]]
      buffer[1000]
      consume]
    , myPort]

let_ Trips2502 = Trips2502a partition["", (.N1 * 25) + .N2, 625]
```

6 Creating and Exporting N-trees at Level 2

Node numbering for export must keep the node numbers for all trees disjoint and node numbers must be consistent with a preorder traversal. We use the following node numbering scheme:

- Root: 0

- 25 level 1 nodes: (.N1 * 250000) + 9999. That is, one less than the start node number of the respective first subtree.
- 625 level 2 root nodes: ((.N1 * 25) + .N2 + 1) * maxSize. As the values for N1 and N2 run from 0 to 24, and maxSize is 10000 (it is necessary to check that the largest tree has less than 10000 nodes), the root nodes of level 2 trees are numbered from 1 * 10000 to 625 * 10000, that is, 10000 to 6250000. So the trees at level 2 must be created with these respective root node numbers and an entry with values N1 and N2 in a level 1 node must have this subtree node number.

In repeated use, we first need to remove previously created tree relations.

```
let_ maxSize = 10000;
query share("maxSize", TRUE, Workers)

let_ Int625 = intstream(0, 624) transformstream
  ddistribute2["", Elem, 625, Workers]
  partition["", .Elem, 0]

query TripsCM200D dcommand['query getcatalog()
  filter[.ObjectName starts "TripsA"] project[ObjectName]
  extend[Deleted: deleteObject(.ObjectName)] count'] filter[.Ok] count;

let_ TripsA = Trips2502 Int625
  areduce2["TripsA", fun(part1: AREDUCEARG1, part2: AREDUCEARG2)
  part1 funseq4[
  $1 feed mconsumeflob,
  $2 mcreatentree7[Trip, 25, 50],
  part2 feed extract[Elem],
  $3 $2 exportntree["TripsA", ($4 + 1) * maxSize, $4]]
  , myPort]
```

7 Collecting Exported Relations into DArrays

Defining relation types:

```
(let_ NodeDist_type = (
  (rel (tuple ((NodeId int) (Entry1 int) (Entry2 int) (Distance real))))
()))

(let_ PivotInfo_type = (
  (rel (tuple ((NodeId int) (Entry int) (PivotDist1 real)
  (PivotDist2 real) (IsPivot bool))))
()))

(let_ TripsNodeInfo_type = (
(rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)
(TotalDistance real) (TotalDuration duration) (SpeedKmh real) (Trip mpoint) (CM
cmpoint) (TID tid) (N1 int) (N1Dist real) (N1x int) (N2 int) (N2Dist real)
(TupleId int) (NodeId int) (Entry int) (Subtree int) (MaxDist real))))
()))
```

Creating darrays over relations on workers. Note that TupleIds created for the local N-trees need to be converted into global TupleIds.

```
let_ NSlots = 625;

let_ TripsAZNodeDist = Workers feed
  createDArray["TripsANodeDist", NSlots, NodeDist_type, TRUE]

let_ TripsAZPivotInfo = Workers feed
  createDArray["TripsAPivotInfo", NSlots,
  PivotInfo_type, TRUE]

let_ TripsAZNodeInfo = Workers feed
  createDArray["TripsANodeInfo", NSlots,
  TripsNodeInfo_type, TRUE]
  dmap["", . feed replaceAttr[TupleId: tid2int(.TID)] consume]
```

8 Computing Tree Relations for the Root Node

Note that it is required that in the node distance relations we have pairs with *Entry1* > *Entry2*.

```
let_ RootNodeDist = Centers25 feed Centers25 feed {c2} product
  filter[.N1 > .N1_c2]
  projectextend[; NodeId: 0, Entry1: .N1, Entry2: .N1_c2,
    Distance: distanceAvg(.Trip, .Trip_c2)]
  sort
  consume

let_ MaxDistancesRoot = Trips2501 dmap["", . feed sortby[N1]
  groupby[N1; MaxDist: group feed max[N1Dist]]]
  dsummarize
  sortby[N1] groupby[N1; MaxDist: group feed max[MaxDist]]
  consume
```

We compare relation types to determine which attributes need to be added to *Centers25* to make schemas equal.

NodeIds must be assigned consistent with a preorder traversal of the tree; there may be gaps between node numbers. We assign node numbers for level 1 nodes as 1 less than the first node number of their subtrees. For *maxSize* = 10000 this means (.N1 * 250000) + 9999.

```
query Centers25 getTypeNL

(rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)
(TotalDistance real) (TotalDuration duration) (SpeedKmh real) (Trip mpoint) (CM
cmpoint) (TID tid) (N1 int))))

query TripsAZNodeInfo getTypeNL

(darray (rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)
(TotalDistance real) (TotalDuration duration) (SpeedKmh real) (Trip mpoint) (CM
cmpoint) (TID tid) (N1 int)

(N1Dist real) (N1x int) (N2 int) (N2Dist real)
(TupleId int) (NodeId int) (Entry int) (Subtree int) (MaxDist real)))))
(N1Dist real) (N1x int) (N2 int) (N2Dist real) (TupleId int) (NodeId int)
(Entry int) (Subtree int) (MaxDist real)))))
```

```
let_ RootNodeInfo = Centers25 feed extend[N1Dist: 0.0, N1x: 0, N2: 0, N2Dist: 0.0,
  TupleId: tid2int(.TID), NodeId: 0, Entry: .N1, Subtree: (.N1 * 250000) + 9999]
  MaxDistancesRoot feed {m}
  itHashJoin[N1, N1_m]
  extend[MaxDist: .MaxDist_m]
  remove[N1_m, MaxDist_m]
  consume

let_ RootPivotInfo =
  Centers25 feed extract[Trip]
  Centers25 feed head[2] tail[1] extract[Trip]
  within2[fun(pivot1: ANY, pivot2: ANY)
    Centers25 feed projectextend[; NodeId: 0, Entry: .N1,
      PivotDist1: distanceAvg(.Trip, pivot1),
      PivotDist2: distanceAvg(.Trip, pivot2),
      IsPivot: .N1 < 2]
    consume]
```

9 Computing Tree Relations for Level 1 Nodes

```
let_ MaxDistancesLevel1 = Trips2502
  areduce["", . feed
  groupby[N1, N2; MaxDist: group feed max[N2Dist]] consume, myPort]
  dsummarize
  consume
```

Comparing relation types:

```
query Centers2501a getTypeNL

(rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)
(TotalDistance real) (TotalDuration duration) (SpeedKmh real) (Trip mpoint) (CM
cmpoint) (TID tid) (N1 int) (N1Dist real) (N1x int) (N2 int)

(N1y int))))

query TripsAZNodeInfo getTypeNL

(darray (rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)
(TotalDistance real) (TotalDuration duration) (SpeedKmh real) (Trip mpoint) (CM
cmpoint) (TID tid) (N1 int) (N1Dist real) (N1x int) (N2 int)

(N2Dist real)
(TupleId int) (NodeId int) (Entry int) (Subtree int) (MaxDist real)))))
```

```
let_ Level1NodeInfo = Centers2501a feed filter[(.N1y mod 20) = 0] remove[N1y]
  extend[N2Dist: 0.0, TupleId: tid2int(.TID), NodeId: (.N1 * 250000) + 9999,
    Entry: .N2, Subtree: ((.N1 * 25) + .N2 + 1) * maxSize]
  extend[N1N2: (.N1 * 25) + .N2]
  MaxDistancesLevel1 feed extend[N1N2: (.N1 * 25) + .N2] {m}
  itHashJoin[N1N2, N1N2_m]
  extend[MaxDist: .MaxDist_m]
  remove[N1N2, N1_m, N2_m, N1N2_m, MaxDist_m]
  consume

let_ Level1NodeInfoD = Level1NodeInfo feed project[Trip, N1, NodeId, Entry]
  ddistribute2["", N1, 25, Workers]

let_ Level1NodeDist =  Level1NodeInfoD dmap["",
  . feed . feed {c2} product
  filter[.Entry > .Entry_c2]
  projectextend[NodeId; Entry1: .Entry, Entry2: .Entry_c2,
    Distance: distanceAvg(.Trip, .Trip_c2)]
  sort
  consume]
  dsummarize
  sort
  consume

let_ Level1PivotInfo = Level1NodeInfoD dmap["",
  . feed extract[Trip]
  . feed head[2] tail[1] extract[Trip]
  within2[fun(pivot1: ANY, pivot2: ANY)
    . feed projectextend[NodeId, Entry
      ; PivotDist1: distanceAvg(.Trip, pivot1),
        PivotDist2: distanceAvg(.Trip, pivot2),
        IsPivot: .Entry < 2]
    consume]]
  dsummarize
  consume
```

10 Merging Tree Relations on the Master

```
let_ TripsANodeDist = RootNodeDist feed
  Level1NodeDist feed concat
  TripsAZNodeDist dsummarize concat
  sortby[NodeId, Entry1]
  consume

let_ TripsANodeInfo = RootNodeInfo feed
  Level1NodeInfo feed concat
  TripsAZNodeInfo dsummarize concat
  sortby[NodeId, Entry]
  consume

let_ TripsAPivotInfo = RootPivotInfo feed
  Level1PivotInfo feed concat
  TripsAZPivotInfo dsummarize concat
  sortby[NodeId, Entry]
  consume;
```

Query on a worker:

```
query TripsATreeInfo_43
```

```
    Variant : 7
     Degree : 25
MaxLeafSize : 50
    RelType : (mem (rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime
              instant) (TotalDistance real) (TotalDuration
              duration) (SpeedKmh real) (Trip mpoint) (CM cmpoint)
               (TID tid) (N1 int) (N1Dist real) (N1x int) (N2 int)
               (N2Dist real)))))
     AttrNo : 6
      Geoid : Generic display function used!
Type : geoid
Value: UNDEF
```

```
let_ TripsATreeInfo = [const rel(tuple([
  Variant: int,
  Degree: int,
  MaxLeafSize: int,
  RelType: text,
  AttrNo: int,
  Geoid: geoid
])) value ()]

query TripsATreeInfo inserttuple[7, 25, 50,
'(mem (rel (tuple ((Trip_id int) (PickupTime instant) (DropoffTime instant)' +
'(TotalDistance real)   (TotalDuration duration) (SpeedKmh real) (Trip mpoint)' +
'(CM cmpoint) (TID tid) (N1 int) (N1Dist real) (N1x int) (N2 int)' +
'(N2Dist real)))))',
6, [const geoid value UNDEF]] count
```

11 Importing the N-tree

```
query meminit(10000);

let_ TripsCM200m = TripsCM200R feed mconsumeflob

let_ trip1 = TripsCM200R feed extract[Trip]

let_ TripsCM200m_Trip_ntree = importntree7("TripsA", trip1)

query TripsCM200m_Trip_ntree TripsCM200m mdistRangeN7[trip1, 300.0] count

# result 31

query TripsCM200m_Trip_ntree TripsCM200m mdistRangeN7[trip1, 150.0]
  extend[Distance: distanceAvg(trip1, .Trip)] project[Trip_id, Distance]
  sortby[Distance] consume

# result:
```

```
 Trip_id : 0
Distance : 0
```

```
 Trip_id : 439433
Distance : 69.04194

 Trip_id : 447581
Distance : 104.493017

 Trip_id : 217356
Distance : 122.643083

 Trip_id : 111787
Distance : 134.62606

 Trip_id : 50939
Distance : 147.443478
```

12 Comparison with Constructed N-tree

```
let TripsCM200m_Trip_ntree2 = TripsCM200m mcreatentree7[Trip, 25, 50]

query TripsCM200m_Trip_ntree2 TripsCM200m mdistRangeN7[trip1, 300.0] count

# result 31

query TripsCM200m_Trip_ntree2 TripsCM200m mdistRangeN7[trip1, 150.0]
  extend[Distance: distanceAvg(trip1, .Trip)] project[Trip_id, Distance]
  sortby[Distance] consume
```

```
 Trip_id : 0
Distance : 0

 Trip_id : 439433
Distance : 69.04194

 Trip_id : 447581
Distance : 104.493017

 Trip_id : 217356
Distance : 122.643083

 Trip_id : 111787
Distance : 134.62606

 Trip_id : 50939
Distance : 147.443478
```