Accelerating Local Search With PostgreSQL 9.1

Jonathan S. Katz
Exco Ventures
Postgres Open – Sep 16, 2011
Local Search?

• Not necessarily location based on places
• “How close are two entities to one another?”
• “What are the closest entities to me?”
• “What are my nearest neighbors?”
Nearest Neighbor Overview

• Want to know “how similar” objects are relative to each other
  – What are the top “k” choices near me?

• Need to define a “metric” for similarity
  – “distance”
K-Nearest Neighbor

• Given a collection of $n$ objects

• When trying to classify an unknown object
  – compute the distance between all known objects
  – find the $k$ ($k \geq 1$) closest objects to the unknown object
  – classify the object based on class of $k$ closest objects

• When $k=1$, then unknown object is given same classification as object it is closest to
K=1 Example

Voronoi Diagram of order 1 can be used to make k=1 NN queries
Applications

- Geolocation + Optimizing Positioning
- Classification
- Similarity
- Recommendation systems
- Content-based image retrieval
- etc.
So what about PostgreSQL?

- As of PostgreSQL 9.0
  - supports geometric types and distances
    - Points, circles, lines, boxes, polygons
    - Distance operator: <->
  - pg_trgm – supplied module for determining text similarity
    - similarity(“abc”, “ade”) computes similarity score
    - <-> defines distance (opposite if similarity), not defined (in 9.0)
PostgreSQL 9.1: KNN-GiST

• Can index data that provides a “<->” (distance) operator
  – Geometric
  – pg_trgm
• “k” = LIMIT
• Known inefficiencies when k=n and n is small
Example: pg_trgm

• Data:
  – List of 1,000,000 names – 700,000 unique

• Indexes:
  – CREATE INDEX names_name_idx ON names (name)
  – CREATE INDEX trgm_idx ON names USING gist (name gist_trgm_ops)

• k=10

• Displaying query plan / execution time after 10 runs
EXPLAIN ANALYZE
SELECT name, similarity(name, 'jon') AS sim
FROM names
WHERE name % 'jon'
ORDER BY sim DESC
LIMIT 10;
pg_trgm: 9.0

Limit (cost=2724.95..2724.98 rows=10 width=14) (actual time=192.793..192.794 rows=10 loops=1)
  ->  Sort (cost=2724.95..2727.45 rows=1000 width=14) (actual time=192.790..192.791 rows=10 loops=1)
      Sort Key: (similarity(name, 'jon '::text))
      Sort Method: top-N heapsort  Memory: 25kB
      ->  Bitmap Heap Scan on names (cost=56.47..2703.34 rows=1000 width=14) (actual time=188.836..192.499 rows=865 loops=1)
          Recheck Cond: (name % 'jon '::text)
          ->  Bitmap Index Scan on trgm_idx (cost=0.00..56.22 rows=1000 width=0) (actual time=188.652..188.652 rows=865 loops=1)
              Index Cond: (name % 'jon '::text)
Total runtime: 192.881 ms
EXPLAIN ANALYZE

SELECT name, similarity(name, 'jon') AS sim
FROM names
WHERE name % 'jon'
ORDER BY sim DESC
LIMIT 10;
pg_trgm 9.1

Limit (cost=2720.91..2720.93 rows=10 width=14) (actual time=202.022..202.023 rows=10 loops=1)
  ->  Sort (cost=2720.91..2723.41 rows=1000 width=14) (actual time=202.020..202.021 rows=10 loops=1)
      Sort Key: (similarity(name, 'jon '::text))
      Sort Method: top-N heapsort  Memory: 25kB
      ->  Bitmap Heap Scan on names (cost=52.43..2699.30 rows=1000 width=14) (actual time=198.324..201.719 rows=865 loops=1)
          Recheck Cond: (name % 'jon '::text)
          ->  Bitmap Index Scan on names_trgm_idx (cost=0.00..52.18 rows=1000 width=0) (actual time=198.156..198.156 rows=865 loops=1)
              Index Cond: (name % 'jon '::text)

Total runtime: 202.113 ms
Comparable?

- Seems to be similar
  - Need to do more research why
- However, 9.1 offers improvements for LIKE/ILIKE search with pg_trgm
LIKE/ILIKE

EXPLAIN ANALYZE

SELECT name
FROM names
WHERE name LIKE '%ata%n';
LIKE/ILIKE pg_trgm: 9.0 vs 9.1

Seq Scan on names
(cost=0.00..18717.00 rows=99 width=14) (actual time=0.339..205.659 rows=665 loops=1)
Filter: (name ~~~ '%ata%n'::text)
Total runtime: 205.743 ms

Bitmap Heap Scan on names
(cost=9.45..369.20 rows=99 width=14) (actual time=122.494..125.967 rows=665 loops=1)
Recheck Cond: (name ~~~ '%ata%n'::text)
-> Bitmap Index Scan on names_trgm_idx
(cost=0.00..9.42 rows=99 width=0) (actual time=121.972..121.972 rows=3551 loops=1)
Index Cond: (name ~~~ '%ata%n'::text)
Total runtime: 126.065 ms
Geometry

• Data:
  – 2,000,000 points, from (0,0) -> (10000, 10000)

• Index:
  – CREATE INDEX geoloc_coord_idx ON geoloc USING gist (coord);
Geometry

EXPLAIN ANALYZE

SELECT *, coord <-> point(500,500)
FROM geoloc
ORDER BY coord <-> point(500,500)
LIMIT 10;
Geometry: 9.0 vs 9.1

Limit (cost=80958.28..80958.31 rows=10 width=20) (actual
time=1035.313..1035.316 rows=10 loops=1)
-> **Sort** (cost=80958.28..85958.28 rows=2000000 width=20) (actual
time=1035.312..1035.314 rows=10 loops=1)
   Sort Key: ((coord '<- (500,500)'::point))
   **Sort Method**: top-N heapsort
   **Memory**: 25kB
   -> **Seq Scan** on geoloc
      (cost=0.00..37739.00 rows=2000000 width=20) (actual
time=0.029..569.501 rows=2000000 loops=1)

  *Total runtime: 1035.349 ms*

Limit (cost=0.00..0.81 rows=10 width=20) (actual time=0.576..1.255
rows=10 loops=1)
-> **Index Scan using geoloc_coord_idx on geoloc** (cost=0.00..162068.96
rows=2000000 width=20) (actual
time=0.575..1.251 rows=10 loops=1)
   Order By: (coord '<- (500,500)'::point)

  *Total runtime: 1.391 ms*
Conclusions

• GiST: “Generalized Search Tree” – index is there, up to developers to define access methods of data types
  – e.g. yields KNN-GiST

• Different types of applications can be built – performance enhancements

• Next steps?
My Wish List

• Further geometric-type support in Postgres
  – N-dimensional points
  – ‘=‘ operator for point type
  – (PostGIS still champion of complex geometric + geographic data types)

• Define “distance” over multicolumns with different types?
  – SELECT (a.name, a.geocode) <-> (b.name, b.geocode) FROM x a, x b;
References


• Oleg Bartunov and Teodor Sigaev for work on KNN-GiST and notes on pg_trgm ([http://developer.postgresql.org/pgdocs/postgres/pgtrgm.html](http://developer.postgresql.org/pgdocs/postgres/pgtrgm.html))

Contact

- Jonathan S. Katz
- jonathan.katz@excoventures.com
- @jkatz05