Scalable Similarity Search on Twitter using Parallel Locality Sensitive Hashing

Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, Pradeep Dubey
Near(est) Neighbor

Definition:
- Given: a set of n points in d dimensions
- Nearest Neighbor: for any query q returns a data point minimizing distance from q
- r-Near Neighbor: for any query q returns a point within distance r from q (if it exists)

Basic geometric problem
- Many applications, especially when the dimension d is high
  - Information retrieval
  - Computer vision
  - Data mining
  - Signal processing
  - ....
Similarity Search on Tweets

Text Similarity can be measured by Angular Distance Similarity of word vectors
Similarity Search on Tweets

Applications:
- Topic detection
- Sentiment analysis
- Similar users
- Hash tag recommendation
Big Data Challenge

- Large number of tweets to search:
  - billions of tweets
- High dimensional data (large vocabulary):
  - > 50000 words
- Dynamic data:
  - ~ 300M tweets/day
Meeting the Performance Challenge

• Existing Solutions:
  ◦ Don’t scale well with large number of points
  ◦ Curse of Dimensionality
    • E.g. k-d trees exhibit running times exponential in \( d \)

• Locality Sensitive Hashing (LSH) [Indyk-Motwani’98, Andoni-Indyk’06’08]:
  ◦ Randomized: reports the neighbor within distance \( r \) with probability \( (1-\delta) \)
  ◦ Query time: \( O(dn^\rho) \)
  ◦ Can be parallelized

• Project goal: highly scalable parallel LSH
LSH idea

- Hash points into buckets, such that for any points $p, q$, the probability that $p, q$ collide depends on the distance between $p$ and $q$:
  - Similar items are more likely to collide

![Diagram of LSH idea with colored points and a circle indicating collision zone]
**LSH details**

- **Preprocessing**: hash points into L hash tables:
  - Each table has k-bit keys
  - Need k*L hash functions, each providing 1 bit (use 1-bit quantized dot product with a random hyperplane)
  - Choose k and L to have a high accuracy 1-δ

- **Query**:
  - For each query q, look only in buckets that q hashes to
  - Calculate angular similarity to select points within distance r
Parallel Implementation of LSH

- **Hash table construction (Preprocessing)**
  - Parallelism over input tweets
  - Use SIMD to speedup hash computations
  - Heavy reuse of hyperplane data due to biased word frequency distribution (good cache behavior)
  - Use architecture-friendly 2-level partitioning scheme to improve locality of hash table insertions and minimize TLB and cache misses

- **Querying**
  - Parallelism over queries
  - Group queries to process them in small batches – increases latency, improves throughput

- **Distance-based filtering**
  - Eliminate duplicates among possible candidates using bit-vectors.
### Results (12M tweets)

**LSH vs Naïve Search**

<table>
<thead>
<tr>
<th></th>
<th>Time in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preprocessing</strong></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>99612.82</td>
</tr>
<tr>
<td>16 Threads (8-cores, SMT)</td>
<td>15094.13</td>
</tr>
<tr>
<td><strong>Query (1000 queries)</strong></td>
<td></td>
</tr>
<tr>
<td>1 Thread</td>
<td>21334.73</td>
</tr>
<tr>
<td>16 Threads (8-cores, SMT)</td>
<td>2836.51</td>
</tr>
</tbody>
</table>

**LSH is 36X faster than naïve (linear) search**

- **LSH multicore scaling**
  - 6.6X on 8 cores for hash table construction
  - 7.5X on 8 cores for querying
PLSH

More results:

- Scaled the experiments to 1 billion tweets (by distributing the data between about 100 nodes)
- Extended to streaming data
- Open access code (available soon)

See poster by Aizana Turmukhametova