DART: Distributed Adaptive Radix Tree for Efficient Affix-based Keyword Search on HPC Systems

Wei Zhang, Houjun Tang, Suren Byna, Yong Chen

November 2nd, 2018
Exponential Data Growth
Mind-blowing Information Explosion
Affix-based Keyword Search

Prefix: AF*

Infix: *FF*

Suffix: *FIX

Database Architecture

Graph Representation
Document-partitioned Approach

I love apple

apple

I love banana

lo*

I love avocado

Query Broadcasting
Term-partitioned Approach - Full String Hashing

I love apple

apple

I love banana

lo*

I love avocado

apple, avocado, banana, I, love

Query Broadcasting

apple

I

avocado

love

banana

...
Term-partitioned Approach - Initial Hashing

I love apple
apple
I love banana
lo*
I love avocado

apple, avocado, banana, I, love

apple, avocado
love

No Query Broadcasting

banana

Load Balance
I
Term-partitioned Approach - Initial Hashing

Imbalance of Keyword Distribution

WIKI
Skewness in Keyword Popularity
Requirements of Distributed Affix-based Keyword Search

- Imbalanced Keyword Distribution
- Skewness of Keyword Popularity
- Avoid Query Broadcasting
  - Document-partitioned Approach
  - Full-String Hashing
- Functionality
- Efficiency
- Load Balance
- Scalability
- Prefix Search
- Suffix Search
- Infix Search
- Exact Search
DART: Distributed Adaptive Radix Tree

### DART Partition Tree Initialization

- Character set $A$, let $k = |A|$ (Radix of DART)
- $M$ = total # of physical machines
- For a partition tree of height $d$, at each level $i \in \{1, \ldots, d\}$, each tree node branches out to level $i + 1$ by iterating each character in the character set $A$ in order.
- Thus, $N_{leaf} = k^d$, and $i_{physical} = \text{leaf} \% M$
- We need to ensure $N_{leaf} \geq M$, thus:
- $d = \lceil \log_k M \rceil + 1$
- Client-side arithmetic calculation.
- $O(1)$ Complexity
- **Root Region** $D_{root} = \frac{N_{leaf}}{k}$ virtual nodes
- **Subregion** $D_{sub} = \frac{N_{leaf}}{k^2}$ virtual nodes
DART: Distributed Adaptive Radix Tree

Index Construction - Overview

- For each term, create index for it and its inverse, e.g., "abc" and "cba"
- Select base virtual node
- Select alternative virtual node
- Select eventual virtual node which has lesser indexed keywords to create the index for the keyword.
- Goal: Balance Keyword Distribution
- Hint: The power of 2-choices
- Randomness can lead to balanced keyword distribution, but will result in query broadcasting.
- Destined keyword placement ensures efficient look up, but leads to imbalanced keyword distribution.

Randomness  Certainty
For term $T = (t_1 t_2 \ldots t_l)$, let $i_{t_n}$ be the index of character $t_n$ in the character set $A$.

- When $l \geq d$, the client calculates:
  
  \[ I_v = \sum_{n=1}^{d} i_{t_n} \times k^{d-n} \]

- E.g. $d = 3$, $A = \{A, B, C\}$, for “CBCBA”

- When $l < d$, the client pad the term with its ending character until $l = d$.

- Perform the above calculation.

- E.g. $d = 3$, $A = \{A, B, C\}$, for “AA”, pad “AA” to “AAA”
DART: Distributed Adaptive Radix Tree

Index Construction – Alternative Virtual Node Selection

- $I_{\text{alter\_region\_start}} = \left[ \left( i_{t_1} + \left\lfloor \frac{k}{2} \right\rfloor \right) \% N_{\text{leaf}} \right] \times D_{\text{root}}$
- E.g. $d = 3, A = \{A, B, C\}$, for “CBCBA”
- $w_1 = (i_{t_{d-1}} + i_{t_d} + i_{t_{d+1}}) \% k$
- $w_2 = \left| (i_{t_{d-1}} - i_{t_d} - i_{t_{d+1}}) \right| \% k$
- $I_v' = I_{\text{alter\_region\_start}} + (I_v + w_1 \times D_{\text{sub}} + w_2) \% D_{\text{root}}$

Randomness
DART: Distributed Adaptive Radix Tree

Index Construction – Eventual Node Selection

- Select node between $I_v$ and $I_{v'}$
- Let $E_v = \begin{cases} I_v, & |I_v| \leq |I_{v'}| \\ I_{v'}, & \text{otherwise} \end{cases}$

Balanced Keyword Distribution
DART: Distributed Adaptive Radix Tree

- To overcome skewness of keyword popularity.
- Replication Factor $r$
- The $i$th replica, $R_i = E_v + \frac{N_{leaf}}{k} \times i, i \in [1, r]$
- E.g. $r = 3$
- Replicas will be accessed in round-robin fashion.

Index Construction – Index Replication

Alleviate Excessive Access on Popular Keywords
DART: Distributed Adaptive Radix Tree

Query Response – Prefix and Suffix Queries

\[ I_{\text{search}} \geq d \text{ for } "\text{CBCB}^*", \]
2 nodes will be accessed.

Base Virtual Node Selection
& Alternative Virtual Node Selection

Access both virtual nodes & Take the result from the node which returns non-empty result
Query Response – Prefix and Suffix Queries

\[ I_{\text{search}} < d, \text{ for } "CB^*" \text{ OR } "C^*", \text{ 2M/k nodes will be scanned} \]
DART: Distributed Adaptive Radix Tree

Query Response – Infix Query

• The position of a given infix is uncertain in a keyword.
• Query broadcasting is inevitable.
• To the best of our knowledge, there is no indexing technique that can avoid full scan on the indexed keywords when it comes to infix query.
DART: Distributed Adaptive Radix Tree

Complexity of DART Operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Computation Complexity of Locating Procedure (Worst Case)</th>
<th>Communication Complexity (Worst Case)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Deletion</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Exact Search</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Prefix Search</td>
<td>$O(1)$</td>
<td>$O\left(\frac{M}{k}\right)$</td>
</tr>
<tr>
<td>Suffix Search</td>
<td>$O(1)$</td>
<td>$O\left(\frac{M}{k}\right)$</td>
</tr>
<tr>
<td>Infix Search</td>
<td>$O(1)$</td>
<td>$O(M)$</td>
</tr>
</tbody>
</table>
DART: Distributed Adaptive Radix Tree

Experimental Setup

• Platform – Cori @ NERSC (2388 nodes in total)
  • 8 – 512 nodes (1/4 nodes occupied)
  • Half client half server
• Dataset –
  • UUID – generated by libuuid
  • DICT – comprehensive keyword set in natural language
  • WIKI – comprehensive real world queries
• Query –
  • 4-letter prefix/suffix/infix and Exact keyword
    • DART partition tree height ranges from 2 to 3 for 4 - 256 server nodes, given 128 characters in standard ASCII.
DART: Distributed Adaptive Radix Tree

Query Throughput (TPS)

Prefix Query

Suffix Query

Insert

Infix Query

Exact Query

Delete
Latency of DART Operations

Prefix

Suffix

Insert

Exact

Infix

Delete
DART: Distributed Adaptive Radix Tree

Load Balance (Measured by CV)

- Coefficient of Variance (CV)
- “Normalized Standard Deviation”
- Fair measurement for data dispersion regardless of size of the dataset
- $C_v = \frac{\sigma}{\mu}$
- $\sigma$=standard deviation
- $\mu$=mean
DART: Distributed Adaptive Radix Tree

Alleviate Excessive Query Accesses on Popular Keywords
DART: Distributed Adaptive Radix Tree

- **Functionality:** DART enables affix-based keyword search in distributed environment.
- **Efficiency:** DART outperforms full string hashing in terms of search efficiency on prefix search and suffix search.
- **Load Balance:** DART outperforms initial hashing in terms of keyword distribution and generally alleviates excessive query workload on popular keywords.
- **Scalability:** Effective on different scale.

- DART can be used in many scenarios, such as serving wildcard query in
  - Distributed object-centric storage systems
  - Distributed metadata management system
  - Distributed graph storage systems (properties of property graph)
  - Distributed database for information retrieval and knowledge discovery.
  - ....
Acknowledgement

- This research is supported in part by the National Science Foundation under grant CNS-1338078, IIP-1362134, CCF-1409946, and CCF-1718336.

- This work is supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC0205CH11231. (Project: Proactive Data Containers, Program manager: Dr. Lucy Nowell).

- This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility.