Distributed Adaptive Radix Tree for Efficient Metadata Search on HPC Systems

Wei Zhang[1], Houjun Tang[2], Suren Byna[2], Yong Chen[1],

[1] Texas Tech University, [2] Lawrence Berkeley National Laboratory

***Abstract*—Affix-based keyword search is fundamental to metadata search on HPC systems. While building an inverted index on metadata to facilitate efficient affix-based keyword search is a common practice for standalone databases and desktop file systems, they are often insufficient for high-performance computing (HPC) systems due to the massive amount of data and the distributed nature of the storage. In this poster, we present Distributed Adaptive Radix Tree (DART) which enables scalable and efficient affix-based search. DART maintains a balanced keyword distribution and optimizes for excessive keyword requests dynamically at scale. Our evaluation shows that compared with the "full string hashing" used by the commonly adopted DHT approach, DART achieves up to 55x throughput speedup for prefix and suffix search, and has a comparable throughput for exact and infix search. Also, DART maintains balanced keyword distribution and alleviates excessive query workload on popular keywords.**

I. Introduction

Affix-based keyword search is essential to metadata search on HPC storage systems. Typical queries of affix-based keyword search are "`name=chem*`" and "`date=*/2017`".

While many standalone data structures are available for building inverted index on a single machine, building distributed inverted index remains challenging on HPC systems. One challenge comes from the excessive communication overhead introduced by query broadcasting. The query has to be broadcasted to all participating nodes if inverted index is created by the "Full String Hashing" use case of distributed hash table (DHT). Another challenge comes from the emerging trend of enabling unlimited user-defined tags in storage systems (e.g., SoMeta [1] and TagIt [2]). Unlimited user-defined tags brings unlimited amount of keywords and unpredictable keyword distribution. This would cause imbalanced query workload and hence query contention and poor resource utilization in a distributed system, which may degrade the efficiency of affix-based keyword search. Thus, it is necessary to consider load balance when creating distributed inverted index.

Given the challenges of distributed affix-based keyword search, in this study, we introduce a distributed adaptive radix tree (DART) for efficient affix-based keyword search. Our experimental evaluation shows that DART is able to facilitate efficient affix-based keyword search while maintaining load balance. Some results of this research have also been accepted as a conference publication and will be presented at PACT'18 conference [3].

II. Distributed Adaptive Radix Tree

As shown in Figure 1, the primary component of DART is its partition tree. All leaf nodes of this partition tree serve as virtual nodes for affix-based addressing in distributed systems.

A. DART Initialization

During the initialization of DART, to ensure that the entire search space, namely M physical compute nodes, can be fully covered by all leaf nodes of the partition tree, the number of virtual leaf nodes N_{leaf} should be larger than the number of physical nodes M. For given character set $\mathcal{A}$, where $k = |\mathcal{A}|$, we set the height of DART partition tree d to be:

$$d = \lceil log_k M \rceil + 1 \tag{1}$$

Thus, we can guarantee that $N_{leaf} > M$.

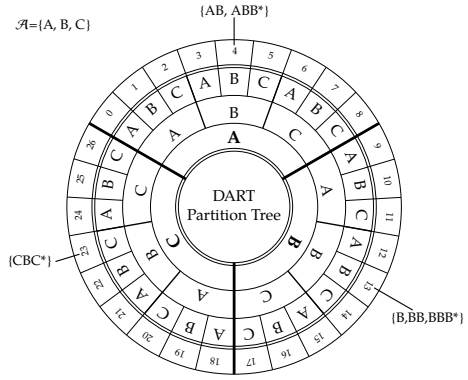


Figure 1: DART partition tree of height $d = 3$ on the basis of character set $\mathcal{A}$ ={A, B, C}. The numbers on the outermost ring are DART virtual leaf node IDs. The search space is divided into k **root regions**, and each region is further divided into k **sub regions**.

B. DART Node Selection Procedures

In DART, all operations require to locate the leaf node based on given string. These operations include index creation, query response, index updating and index deletion.

*1) **Base Virtual Node Selection**:* For a given string $T = (t_1 t_2 ... t_l)$ of character set $\mathcal{A}$, let i_{t_n} denote the index of character t_n in character set $\mathcal{A}$. When the length of string T is greater than or equals to d, namely $l \geq d$, we calculate the base virtual node location I_v by the following equation:

$$I_v = \sum_{n=1}^{d} i_{t_n} \times k^{d-n} \tag{2}$$

When $l < d$, we first pad the term with its ending character until its length reaches d, and then perform the above calculation to determine the virtual node to start from.

*2) **Alternative Virtual Node Selection**:* First, according to the first character of a keyword, we select an alternative root region for the keyword by performing the following calculation:

$$I_{alter\_region\_start} = [(i_{t_1} + \lceil k/2 \rceil) \% N_{leaf}] \times \mathcal{D}_{root} \tag{3}$$

Afterwards, we further calculate the **major offset**, denoted as w_1:

$$w_1 = (i_{t_{d-1}} + i_{t_d} + i_{t_{d+1}}) \% k \tag{4}$$

and then the **minor offset**, denoted as w_2:

$$w_2 = |(i_{t_{d+1}} - i_{t_d} - i_{t_{d-1}})| \% k \tag{5}$$

Finally, we can calculate alternative virtual node ID $I_v{}'$:

$$I_v{}' = I_{alter\_region\_start} + (I_v + w_1 \times \mathcal{D}_{sub} + w_2)\%\mathcal{D}_{root} \quad (6)$$

*3) **Eventual Virtual Node Selection for Index Creation**:* After getting the base virtual node I_v and the alternative virtual node $I_v{}'$ for term T, the client will retrieve the number of indexed keywords on corresponding physical nodes of both virtual nodes, and the client will pick the virtual node with less indexed keywords as the eventual virtual node E_v. The index will be created on virtual node E_v.

*4) **Index Replication**:* To overcome excessive query workload on popular keywords, in DART, we replicate index for r times at virtual nodes R_1 ... R_i, where:

$$R_i = E_v + \frac{N_{leaf}}{k} \times i \quad (7)$$

C. Various Operations on DART

For query response, we simply check both base virtual node and alternative virtual node. We follow round-robin fashion to access the replicas. For update operation and delete operations, we send command to both base virtual node and alternative virtual node, but only the virtual node which owns the given keyword will take action.

III. Evaluation

A. Experimental Setup

On Cori, a Cray XC40 supercomputing system located at the National Energy Research Scientific Computing Center (NERSC), we evaluated DART with three datasets: **UUID**, **DICT** [4], and **WIKI** [5] on standard ASCII character set with 128 characters.

B. Search Throughput at Scale

As shown in Figure 2, for prefix search and suffix search, DART is able to achieve better search throughput (up to 55× higher) than "Full String Hashing" and "Initial Hashing" use cases of DHT. For exact search and infix search, DART is able to achieve comparable search throughput as compared to both DHT use cases.

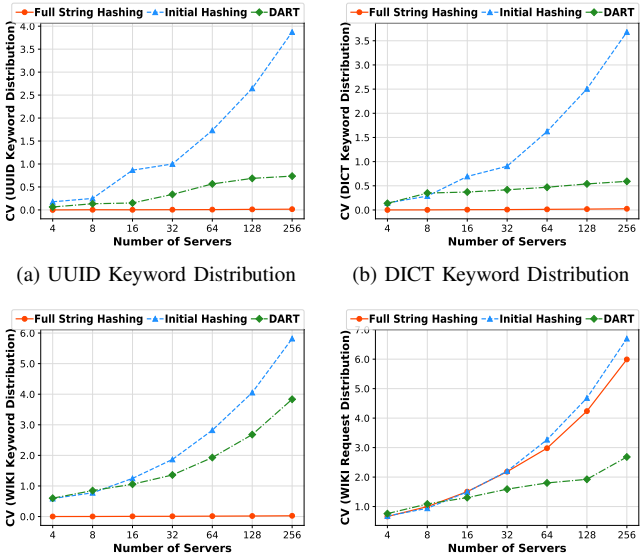
(a) Prefix Search Throughput (b) Suffix Search Throughput

(c) Exact Search Throughput (d) Infix Search Throughput

Figure 2: Search Throughput at Scale

C. Load Balance at Scale

We use Coefficient of Variance (CV) $\mathcal{C}_v = \frac{\sigma}{\mu}$ [6] to measure the dispersion of keyword distribution. As shown in Figure 3, the dispersion of keyword distribution generated by DART is in between both DHT cases, which means DART is able to achieve balanced keyword distribution for all three datasets over different number of servers, although perfect balance is not its goal. Also, the dispersion of request distribution on different keywords generated by DART is also between both DHT cases, which means DART is able to alleviate excessive query workload on popular keywords.



(a) UUID Keyword Distribution (b) DICT Keyword Distribution

(c) WIKI Keyword Distribution (d) WIKI Request Distribution

Figure 3: Load Balance at Scale

IV. Conclusion

With the goal of developing a distributed affix-based keyword search on HPC systems, we have developed a trie-based inverted indexing technique, called DART (Distributed Adaptive Radix Tree). DART can efficiently locate a keyword based on its partition tree and node selection procedures. Our evaluation results show that DART can achieve efficient affix-based keyword search while maintaining load balance at large scale. Particularly, when comparing with full string hashing DHT, the throughput of prefix search and suffix search using DART was up to 55× faster than full string hashing over different number of servers. The throughput of exact search and infix search on DART remains comparable to that of full string hashing. Compared with initial hashing DHT, DART is able to maintain balanced keyword distribution and request distribution while initial hashing fails to achieve. These two advantages of DART are also independent on dataset, making DART an ideal inverted indexing technique for distributed affix-based keyword search. While DHT is prevalently used for data placement in many applications, in scenarios where affix-based search is required or frequent on HPC systems, DART is a competitive replacement of DHT since it provides scalable performance and achieves load balance at scale. In our future work, we plan to apply DART to in-memory metadata object management systems in searching metadata objects using affixes of user-defined tags or other object attributes.

ACKNOWLEDGMENT

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-AC02-05CH11231. (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.

REFERENCES

[1] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol, "SoMeta: Scalable Object-Centric Metadata Management for High Performance Computing," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 359–369.

[2] H. Sim, Y. Kim, S. S. Vazhkudai, G. R. Vallée, S.-H. Lim, and A. R. Butt, "TagIt: An Integrated Indexing and Search Service for File Systems," 2017.

[3] W. Zhang, H. Tang, S. Byna, and Y. Chen, "DART: Distributed Adaptive Radix Tree for Efficient Affix-based Keyword Search on HPC Systems," Accepted to appear in Proceedings of The 27th International Conference on Parallel Architectures and Compilation Techniques (PACT'18), 2018., November 2018.

[4] Marius, "English Words," https://github.com/dwyl/english-words, 2017.

[5] G. Urdaneta, G. Pierre, and M. van Steen. (2009, July) Wikipedia workload analysis for decentralized hosting. http://www.globule.org/publi/WWADH$_comnet2009.html$.

[6] B. Everitt and A. Skrondal, *The Cambridge dictionary of statistics*. Cambridge University Press Cambridge, 2002, vol. 106.