



Distributed Adaptive Radix Tree for Efficient Metadata Search on HPC Systems

Wei Zhang[†], Houjun Tang^{*}, Suren Byna^{*}, Yong Chen[†]



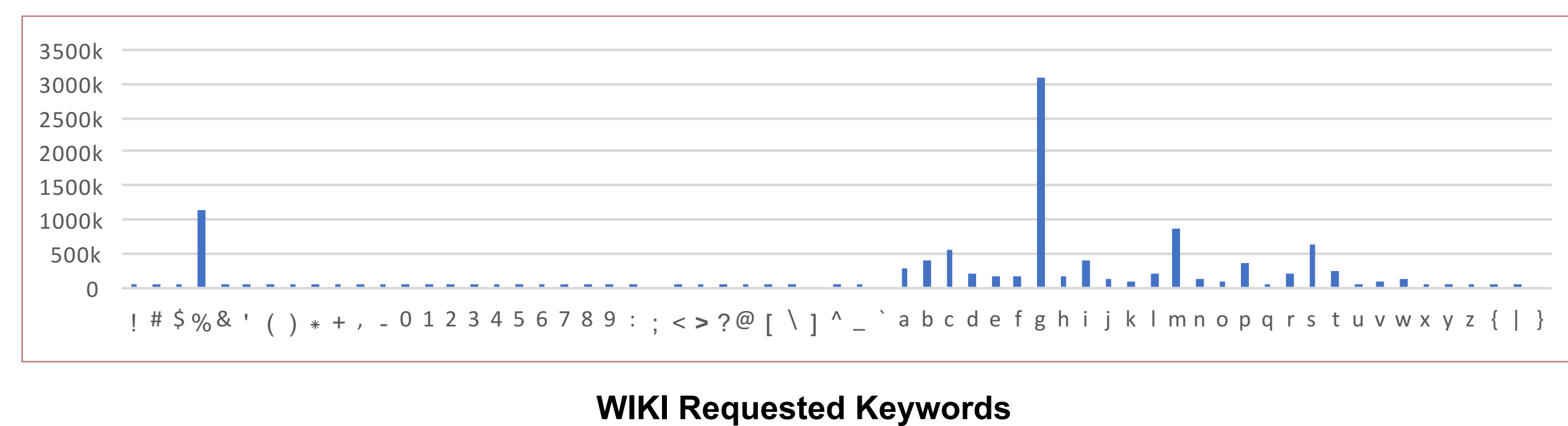
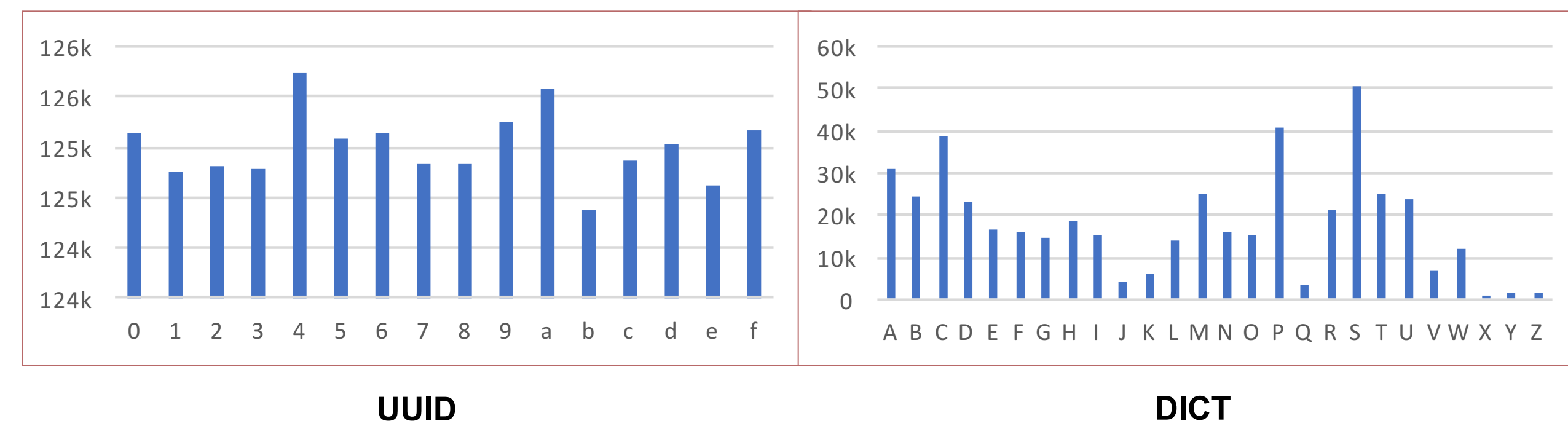
[†]Department of Computer Science, Texas Tech University. ^{*}Lawrence Berkeley National Laboratory.

Abstract

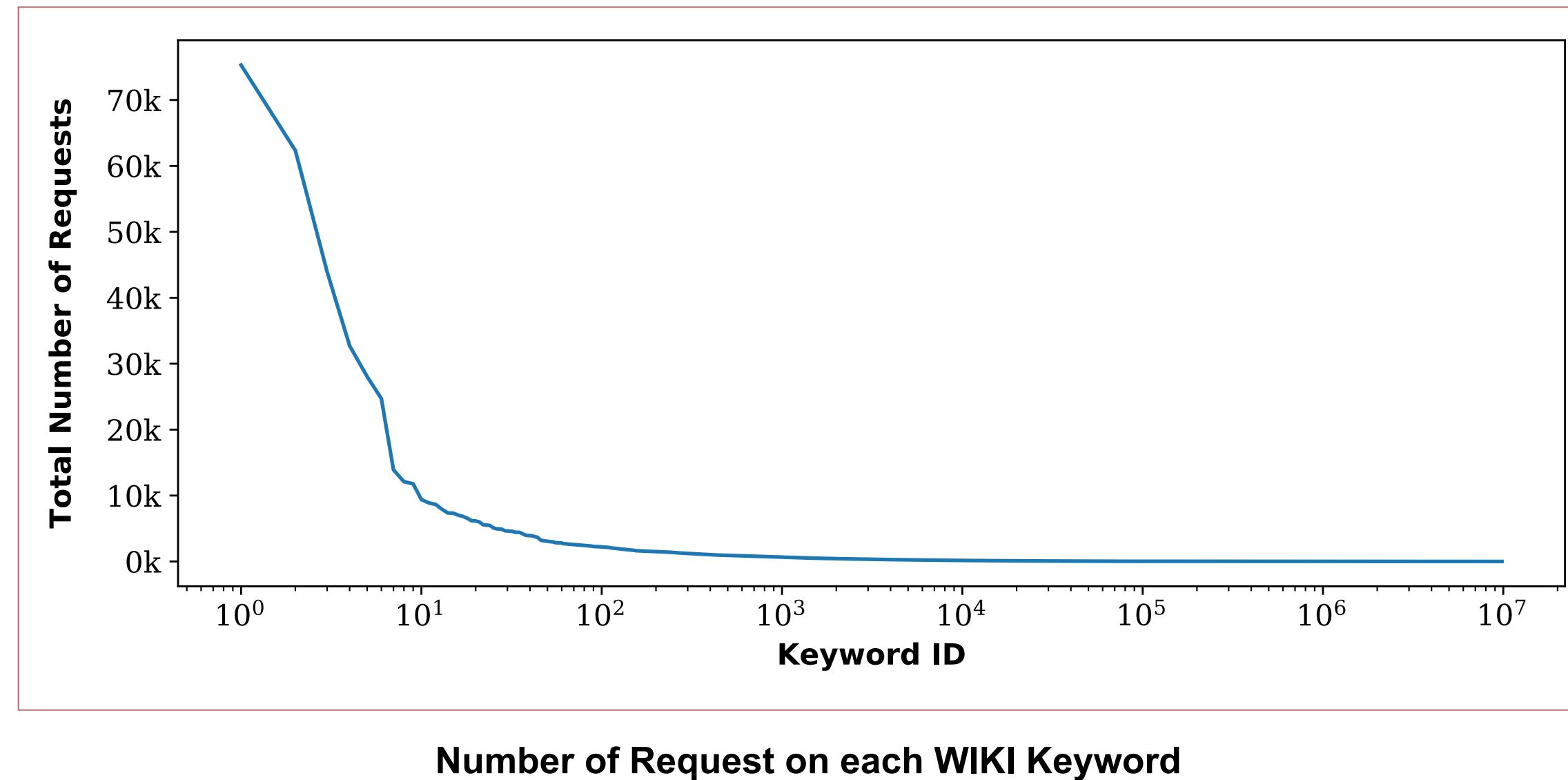
- Affix-based keyword search is fundamental to metadata search on HPC systems.
- Building inverted index to boost the performance of affix-based keyword search is a common practice but is challenging in distributed system like HPC.
- We propose Distributed Adaptive Radix Tree (DART) to address the challenge of building such index in distributed environment.
- Our experimental evaluation exhibits the effectiveness and efficiency of DART.

Background

- Building inverted index for keyword search problem is a common practice.
- In distributed system, building inverted index for keywords involves two different paradigms – document-partitioned approach and term-partitioned approach.
- Document-partitioned approach suffers from heavy overhead introduced by query broadcasting.
- “Full String Hashing” in term-based approach still suffers from query broadcasting in prefix search and suffix search
- “Initial Hashing” in term-based approach suffers from imbalanced workload introduced by uneven keyword distribution led by different leading character.



- Also, uneven query workload on popular keyword is noticeable.



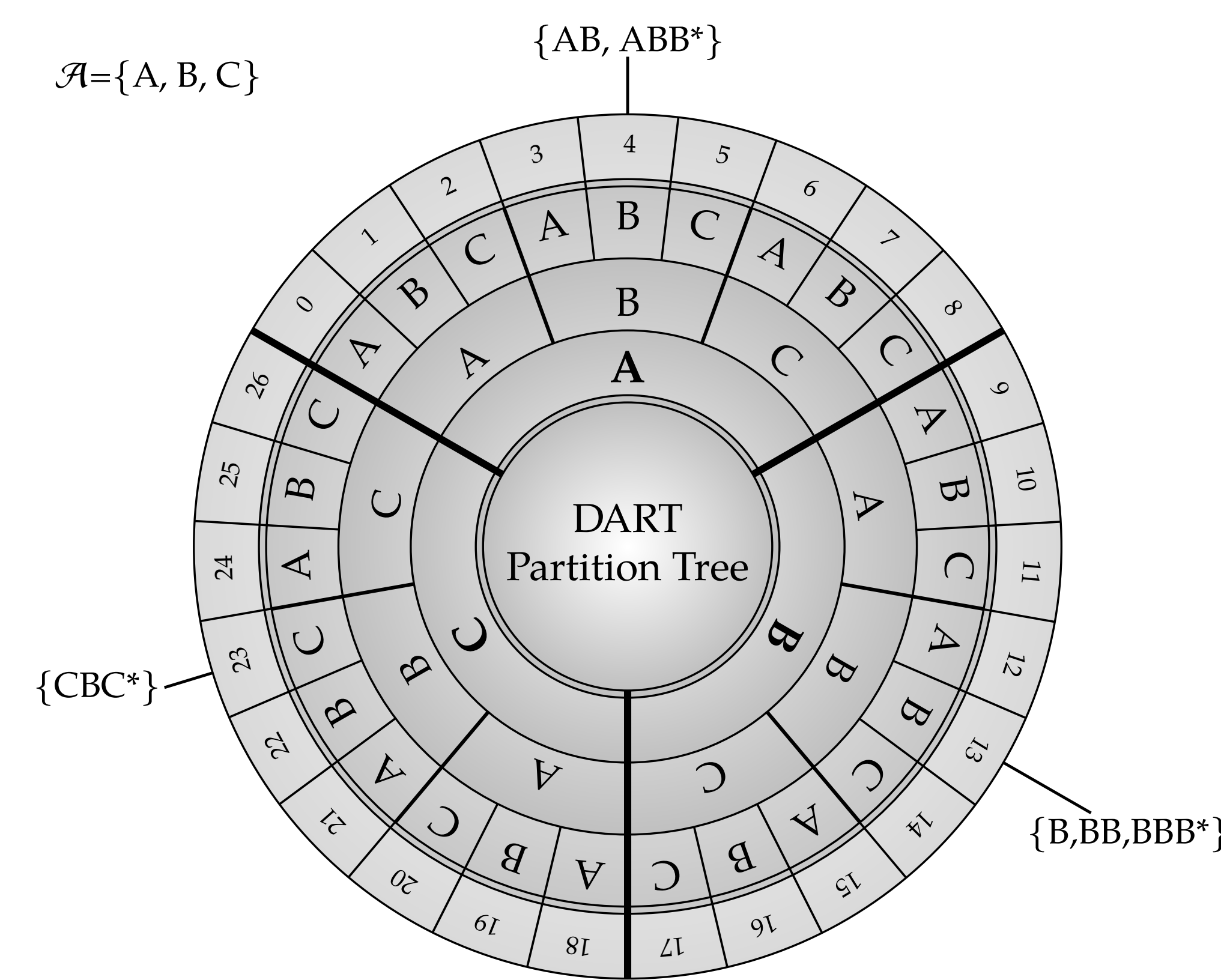
Methods

➤ DART Partition Tree Initialization

While initializing DART on character set with k characters, we have to make sure the total number of leaf nodes in DART is larger than the total number of machines M . Thus, we initialize the height of the DART partition tree d as:

$$d = \lceil \log_k M \rceil + 1$$

Then, we have the following partition tree:



➤ DART Base Virtual Node Selection

Then, for a given string, starting from the root node of the partition tree, we locate the leaf node by following the path containing the sequence of first d characters in the given string. We choose such node as the base virtual node.

➤ DART Alternative Virtual Node Selection

For load balance purpose, we choose an alternative virtual node. We defined a different function for selecting the alternative virtual node.

Keyword	Base Root Region	Base Virtual Node	Alternative Root Region	Alternative Virtual Node
AB	A (begins at 0)	4	C (begins at 18)	21
ABB	A (begins at 0)	4	C (begins at 18)	21
ABBC	A (begins at 0)	4	C (begins at 18)	25
B	B (begins at 9)	13	A (begins at 0)	3
BB	B (begins at 9)	13	A (begins at 0)	3
BBBA	B (begins at 9)	13	A (begins at 0)	3
CBCBA	C (begins at 18)	23	B (begins at 9)	10

➤ DART Eventual Node Selection

After selecting the base virtual node and alternative node, we detect the number of keywords indexed on them, and select the one with less keywords indexed as the eventual node.

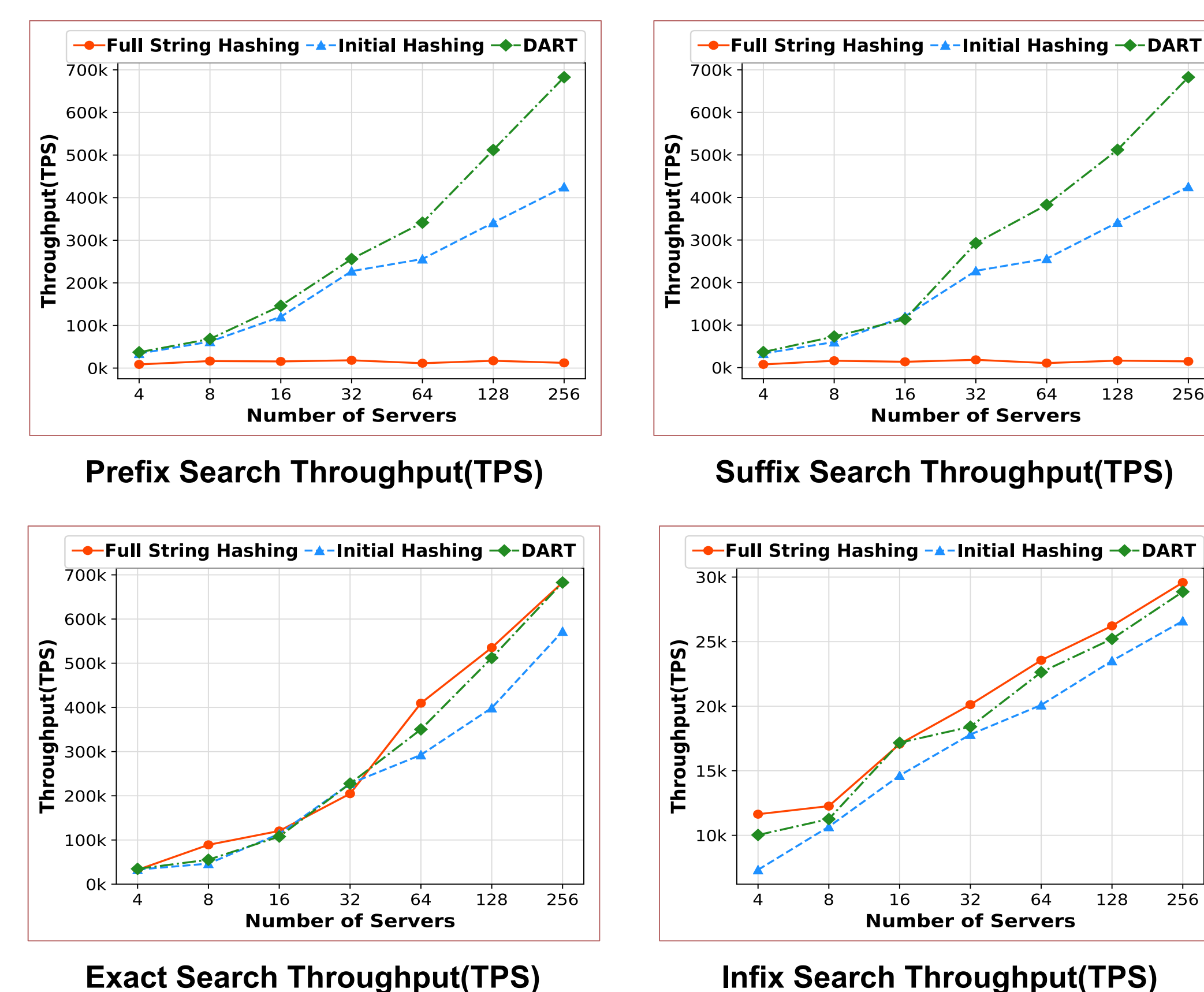
➤ DART Operation Complexity

Operations	Computation Complexity of Locating Procedure (Worst Case)	Communication Complexity (Worst Case)
Insertion	$O(1)$	$O(1)$
Deletion	$O(1)$	$O(1)$
Exact Search	$O(1)$	$O(1)$
Prefix Search	$O(1)$	$O(\frac{M}{k})$
Suffix Search	$O(1)$	$O(\frac{M}{k})$
Infix Search	$O(1)$	$O(M)$

Search Performance at Scale

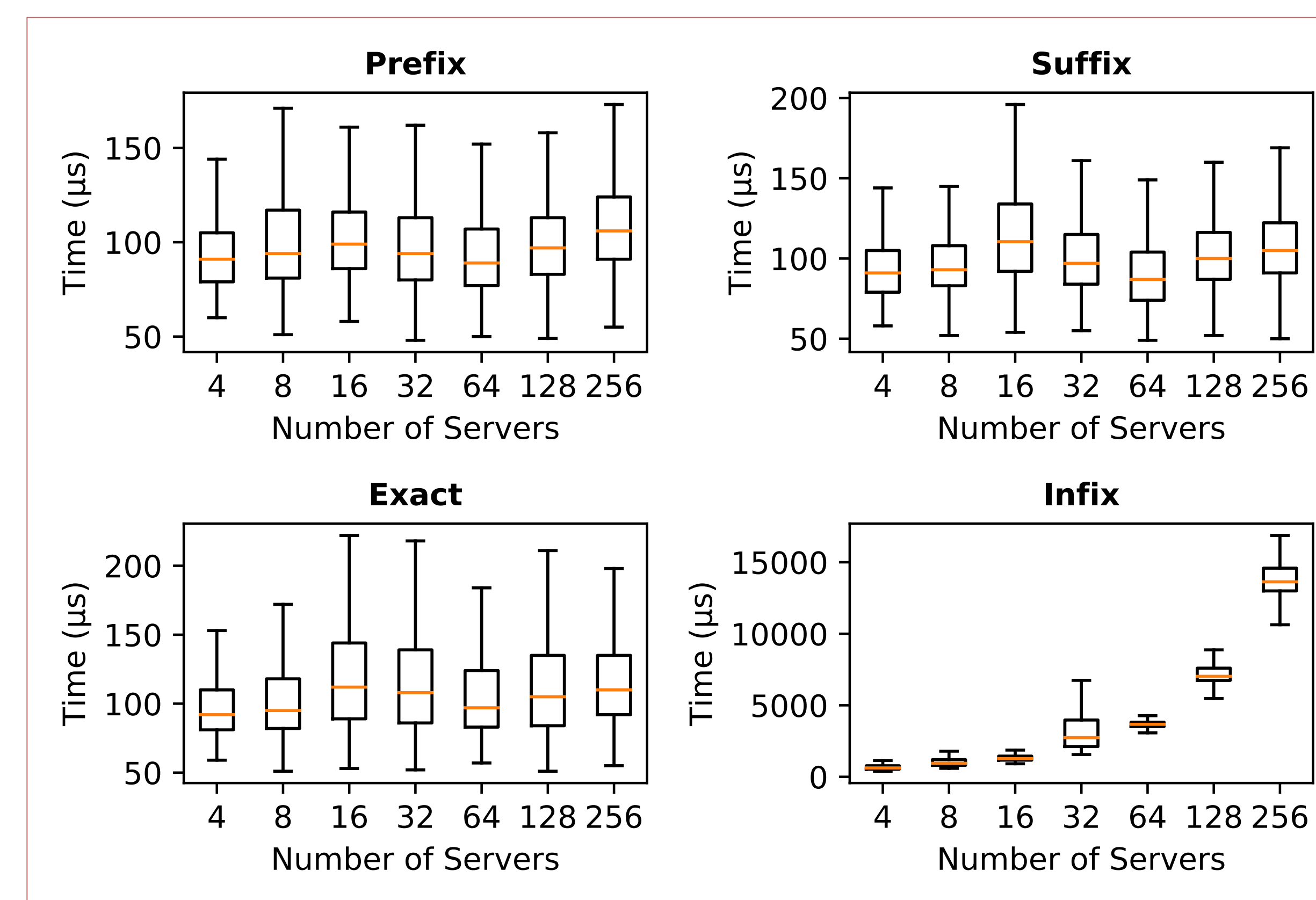
➤ Search Throughput

DART is able to achieve efficient affix-based keyword search at scale. As compared to two DHT cases, DART performs up to 55 times better in prefix search and suffix search while maintaining comparable performance in exact search and infix search.



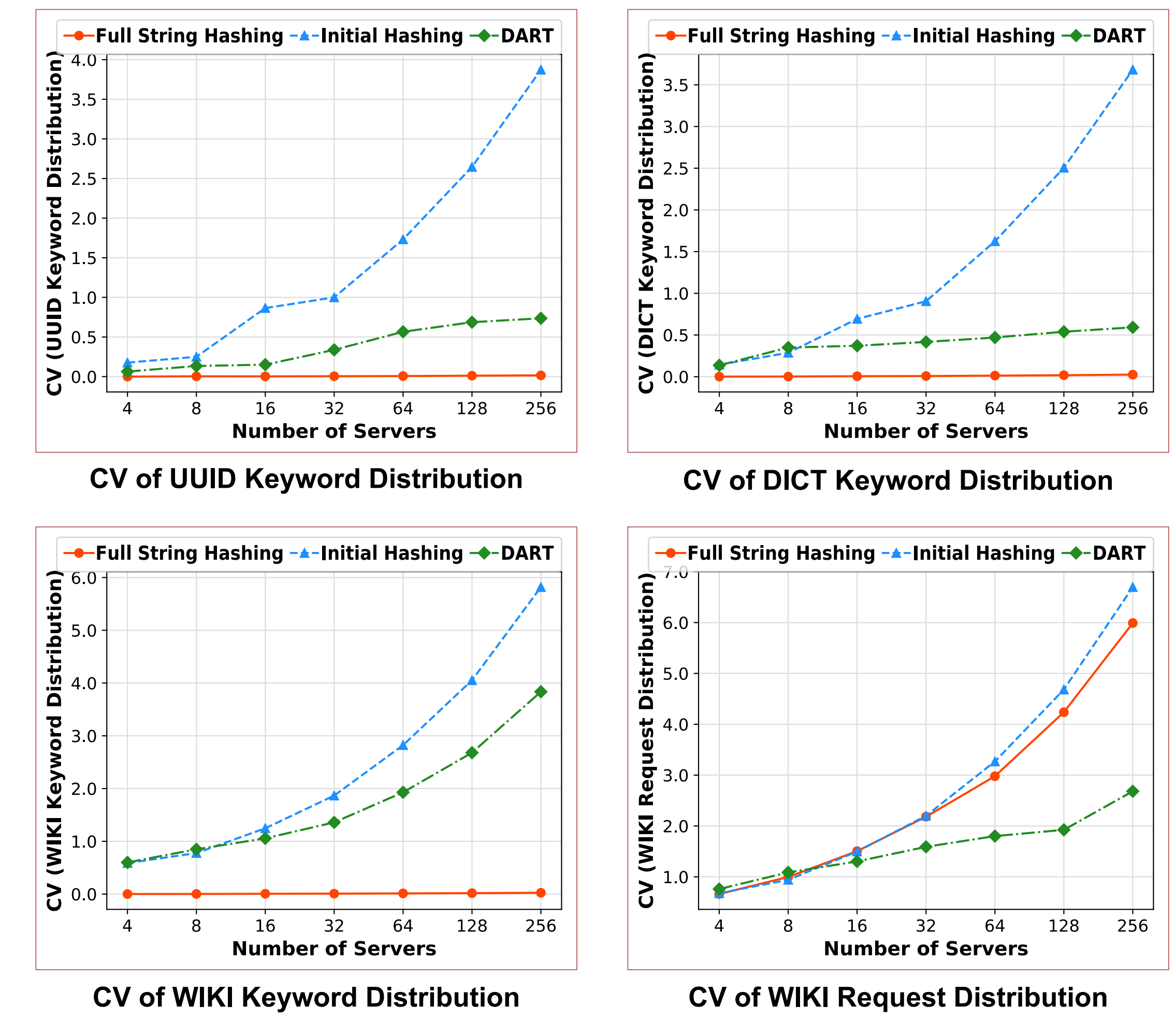
➤ Query Latency

As the number of server increases, DART is able to maintain decent query latency which remains under 250 microseconds for prefix search, suffix search and exact search. For infix query, the query latency on DART still remains within 20 milliseconds on 256 servers, which is the largest scale we have ever tested.



Load Balance at Scale

DART is able to maintain load balance at scale. With different number of servers and different dataset, the dispersion of keyword distribution and query request distribution of DART is always between “full string hashing” and “initial hashing”.



Conclusion

- We have developed a trie-based inverted indexing technique, called DART, by which efficient affix-based keyword search can be performed.
- Our evaluation result shows that DART can successfully facilitate affix-based keyword search and achieve up to 55 times better search performance than “Full String Hashing” DHT use case, while maintaining load balance introduced by imbalanced keyword distribution by different leading characters and uneven query workload of different keyword.

References:

- [1] 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 the Proc. of The 27th International Conference on Parallel Architectures and Compilation Techniques (PACT'18), 2018. (acceptance rate: 36/126=28.6%)
- [2] H. Tang, S. Byna, B. Dong, J. Liu, and Q. Koziol. 2017. SoMeta: Scalable Object-Centric Metadata Management for High Performance Computing. In Cluster Computing (CLUSTER), 2017 IEEE International Conference on. IEEE, 359–369.

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.

