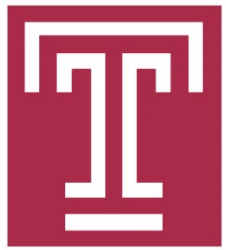
The Dynamic Cuckoo Filter

Hanhua Chen, Liangyi Liao, Hai Jin, Jie Wu

Huazhong University of Science and Technology

Temple University



Two Core Problems

- Set representation
  - organize the information of the elements of a set using some data structure
  - make the information of the set elements operable by corresponding methods
- Set membership testing
  - determine whether an element with a given attribute value belongs to a given set

Emergence of Dynamic Sets

- Real world big data applications
  - Cloud storage
  - Stream applications
- Stringent requirements for dynamic sets
  - Flexibly capacity extending or reducing
  - Reliable delete operation

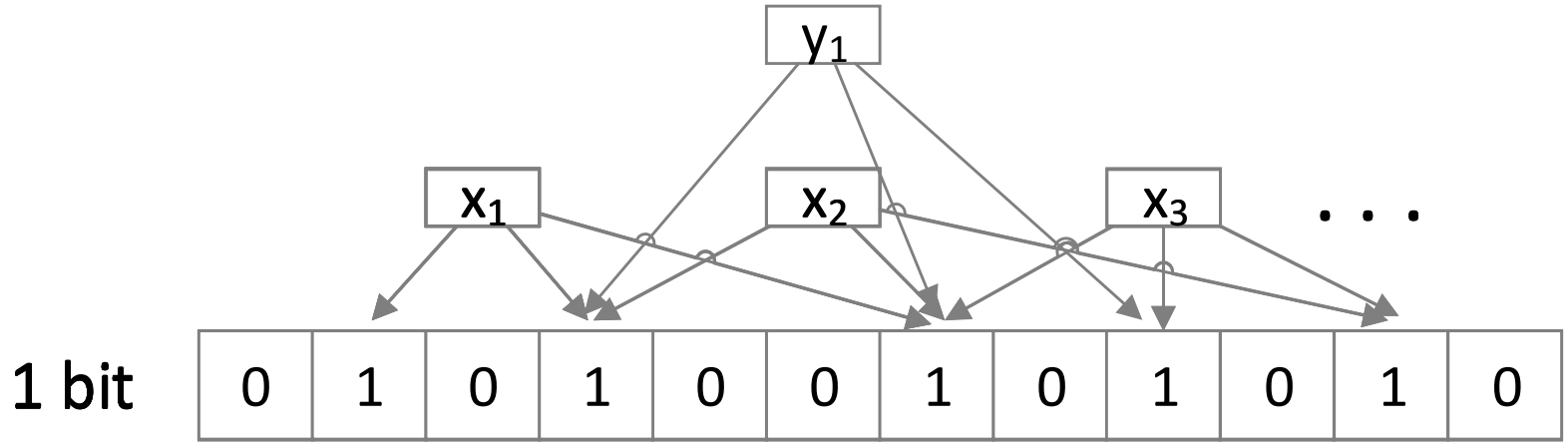
Cloud Storage

Big Stream Data

Existing Designs

Existing Designs	Elastic capacity	Reliable delete	Space cost
Bloom filter	✗	✗	1
Counting Bloom filter	✗	✓	*N*
Cuckoo filter	✗	✓	1
Dynamic Bloom filter	✓	✗	*N*

Bloom filter (BF)

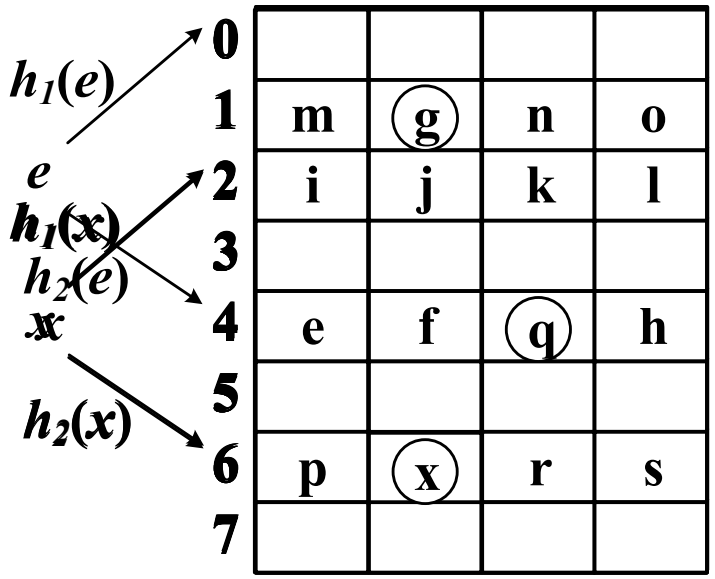


Cuckoo Filter (CF)

Cuckoo Filter (CF):

- *An array of buckets*
- Each bucket has *b entries*
- *Fingerprint* based matching
- *Relocation* when collision

$$h_1(x) = hash(x)$$
$$h_2(x) = h_1(x) \oplus hash(\xi_x)$$

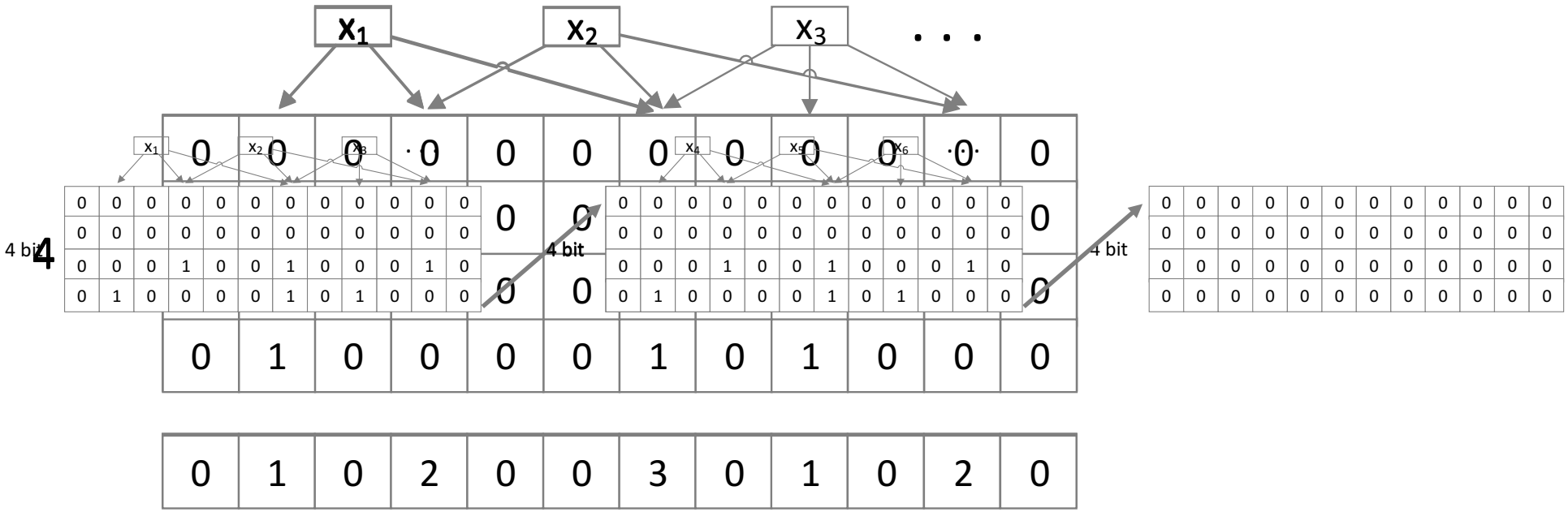


Drawback: *fixed capacity*

"Cuckoo filter: Practically better than bloom," in *CoNEXT,* 2014

Dynamic Bloom Filter

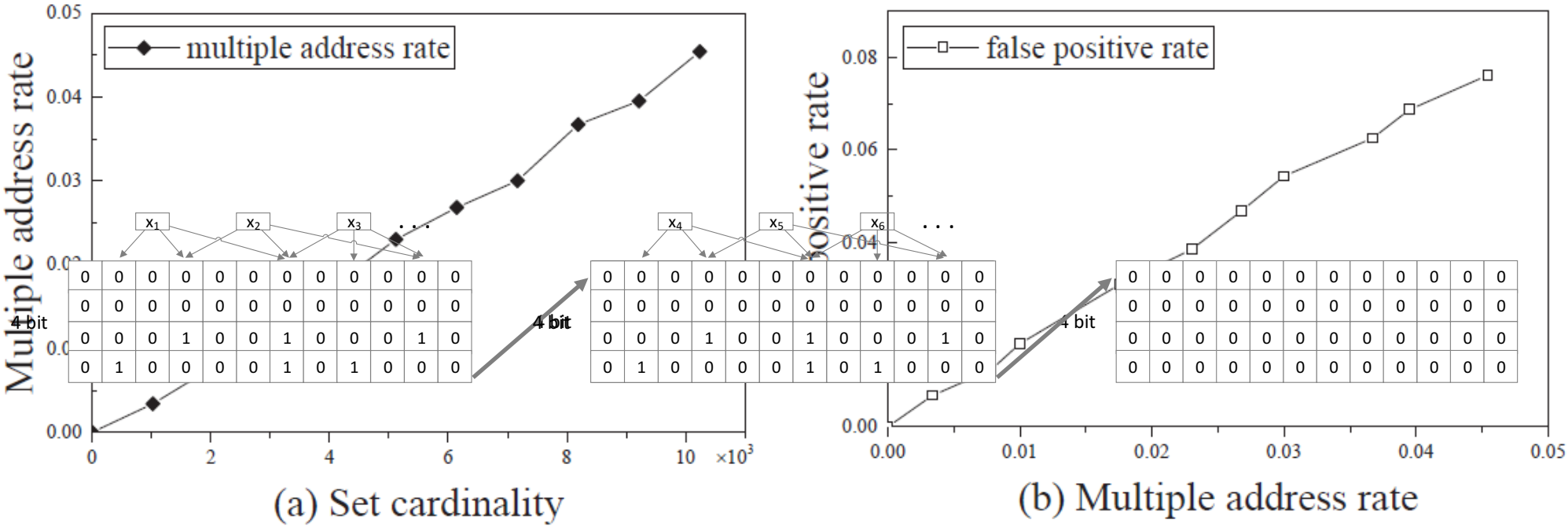
- Linked list of *s* Counting Bloom Filters (CBF)
- Extends capacity by appending new building blocks of CBFs
- Drawback*: unreliable deletion*



"The dynamic bloom filters," *TKDE,* 2010.

Multiple Address Problem in DBF

- Give up deleting the element found in multiple building blocks
- Raises up false positives



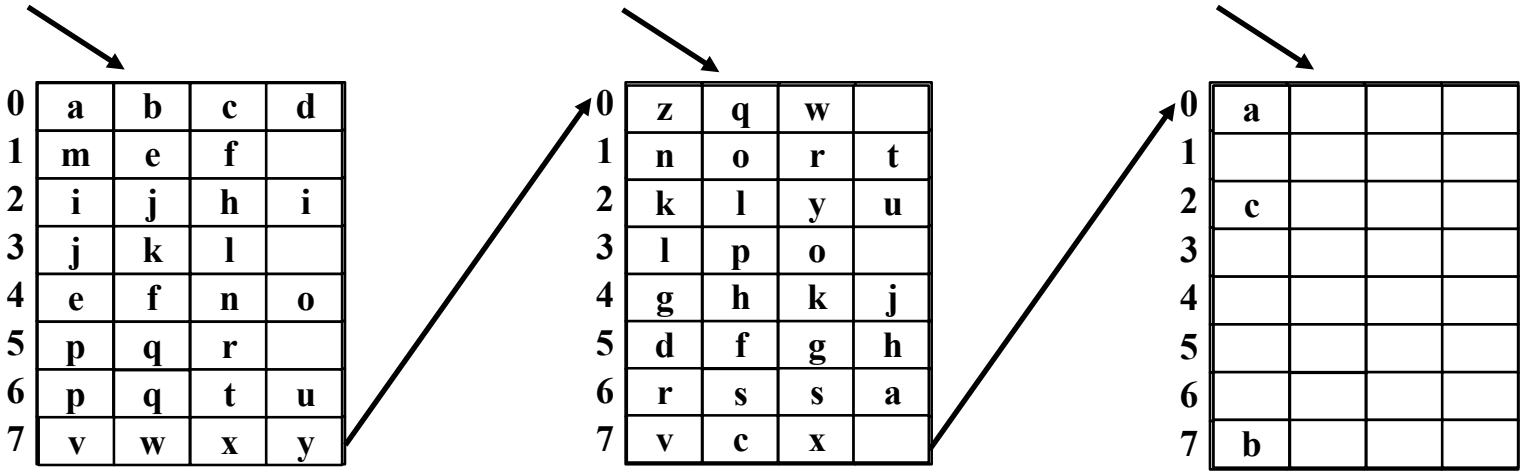
(a) Set cardinality

(b) Multiple address rate

Our Design: Dynamic Cuckoo Filter (DCF)

- DCF uses Cuckoo Filter(CF) as building block
- Extends capacity by appending new building blocks
- Monopolistic fingerprint avoid multiple address problem

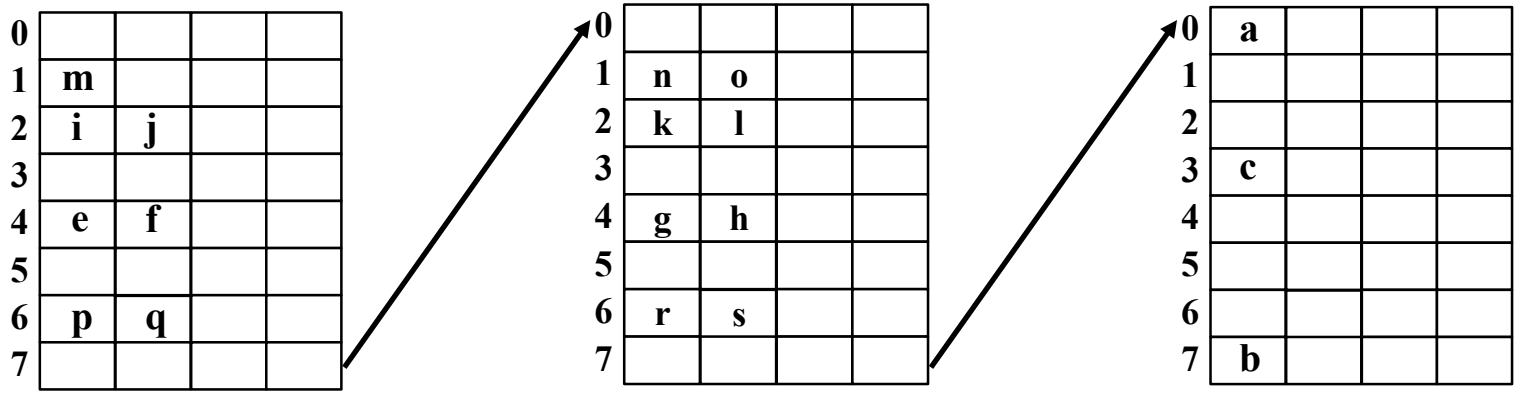
Insert & Query & Deletion



Dynamic Cuckoo Filter (DCF)

Compaction

- Leverage greedy strategy
- Empty a building block with the least fingerprint movements



Testing & Analysis

Dataset	Maximum set cardinality	False positive rate	# of building blocks	Metrics
Random Dataset	46,080	1.17×10^{-2}	6~96	I/Q/D speed

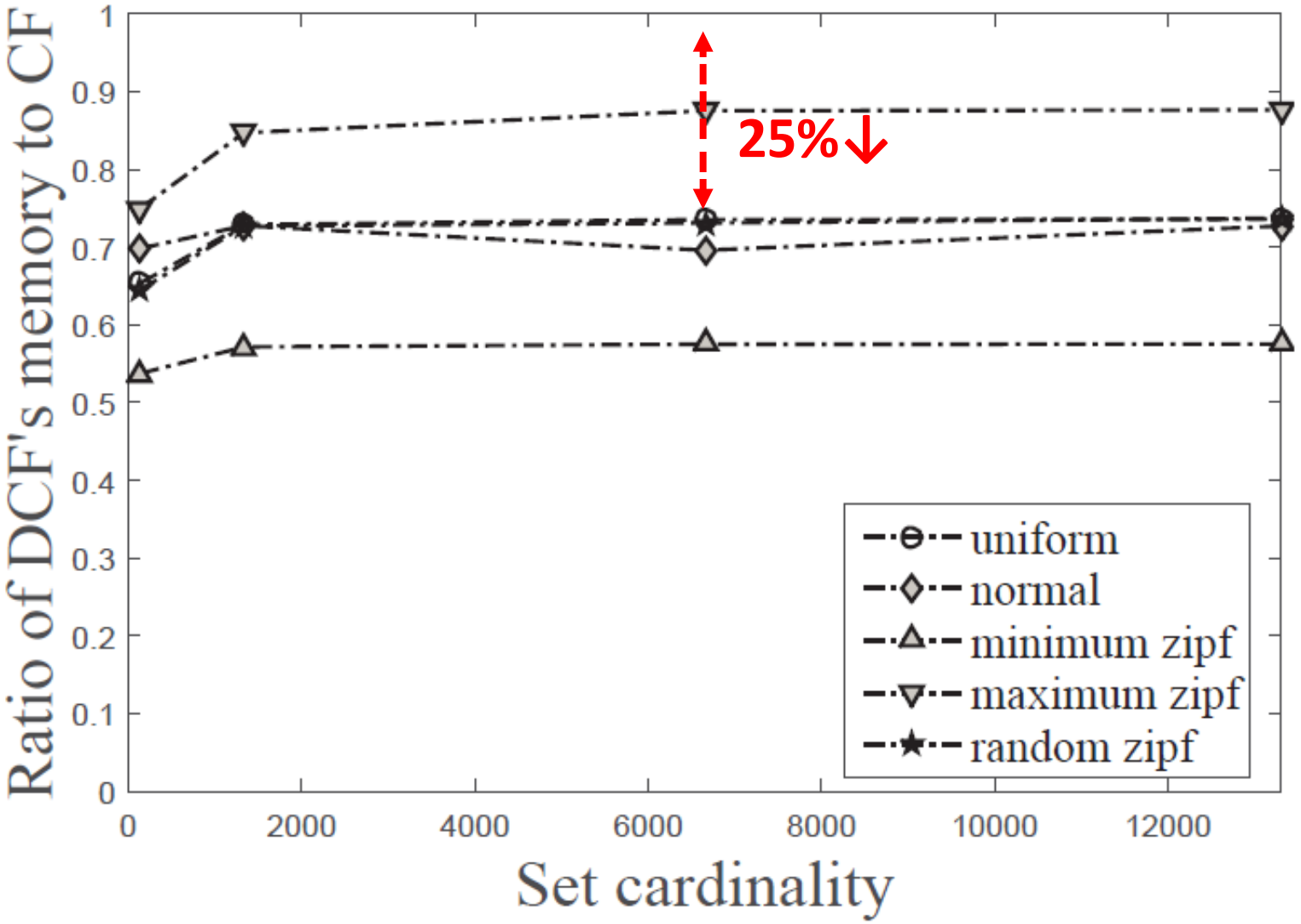
Baseline: the Dynamic Bloom Filter(DBF)

Two sets of experiment

- Varying set cardinality *N* under optimized building block number *s*
- Varying *s* under a fixed set cardinality *N*

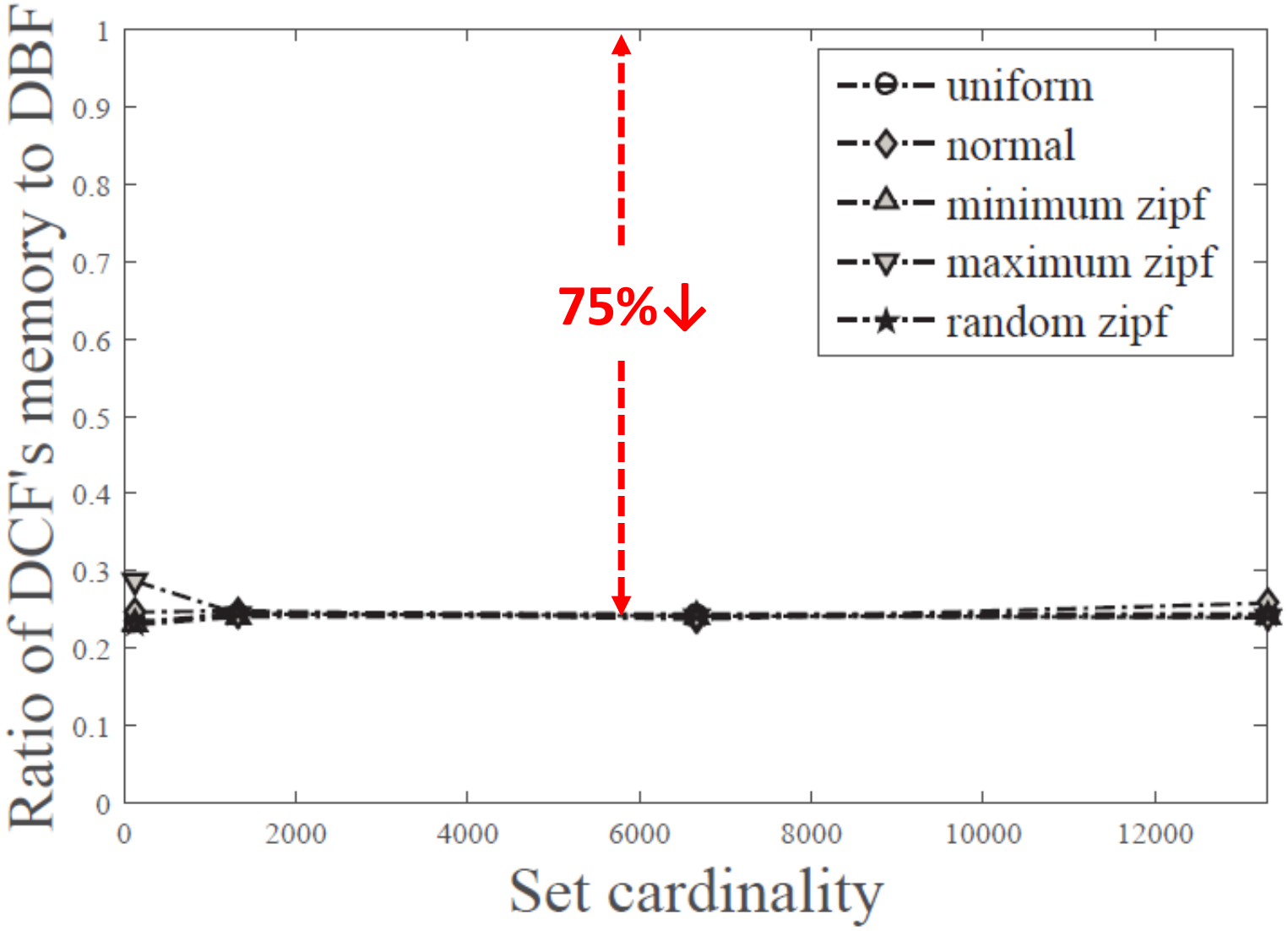
Space Optimization

Space optimization under 5 different set size distribution

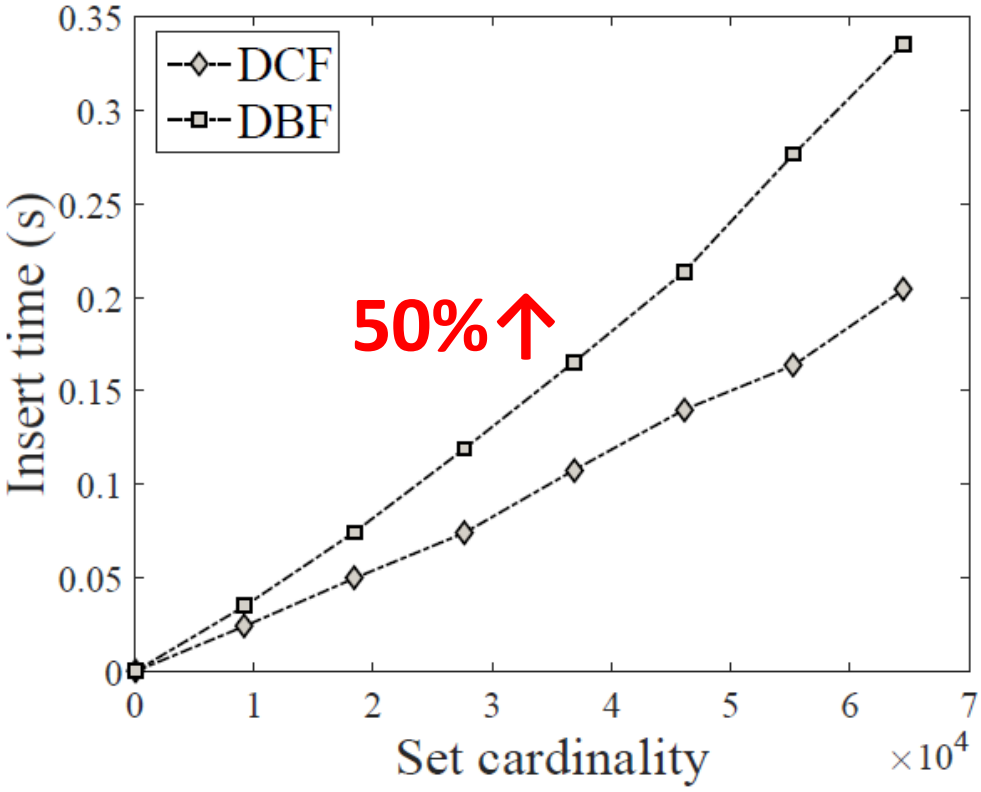


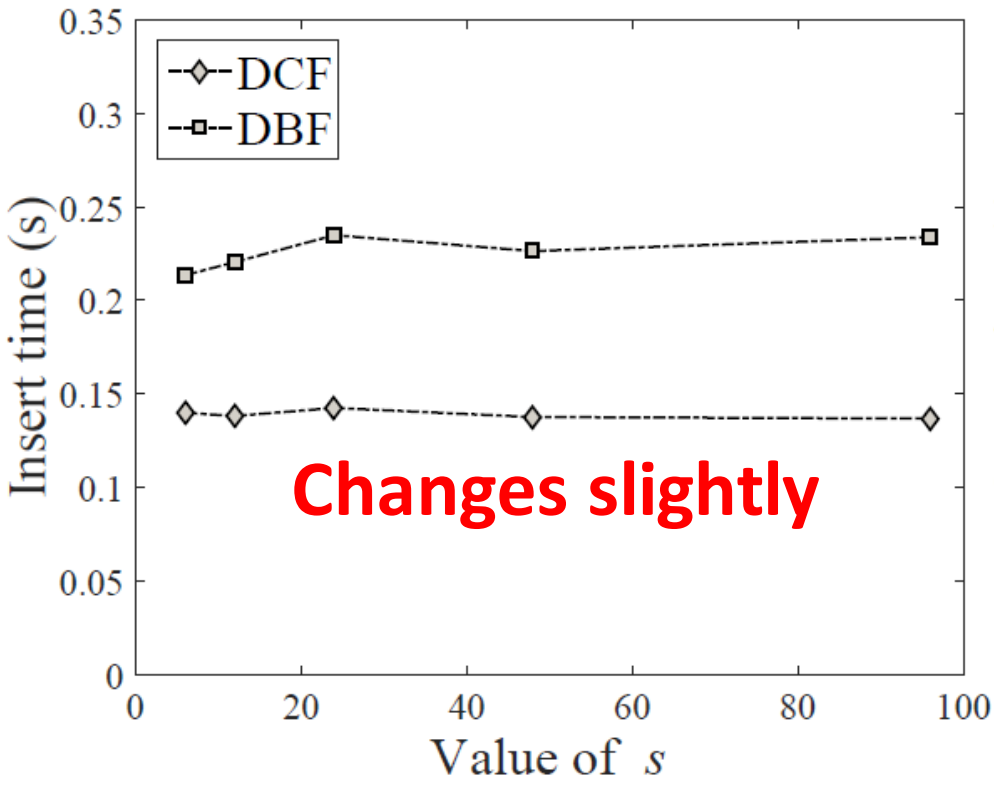
Space Optimization

Space optimization under 5 different set size distribution:

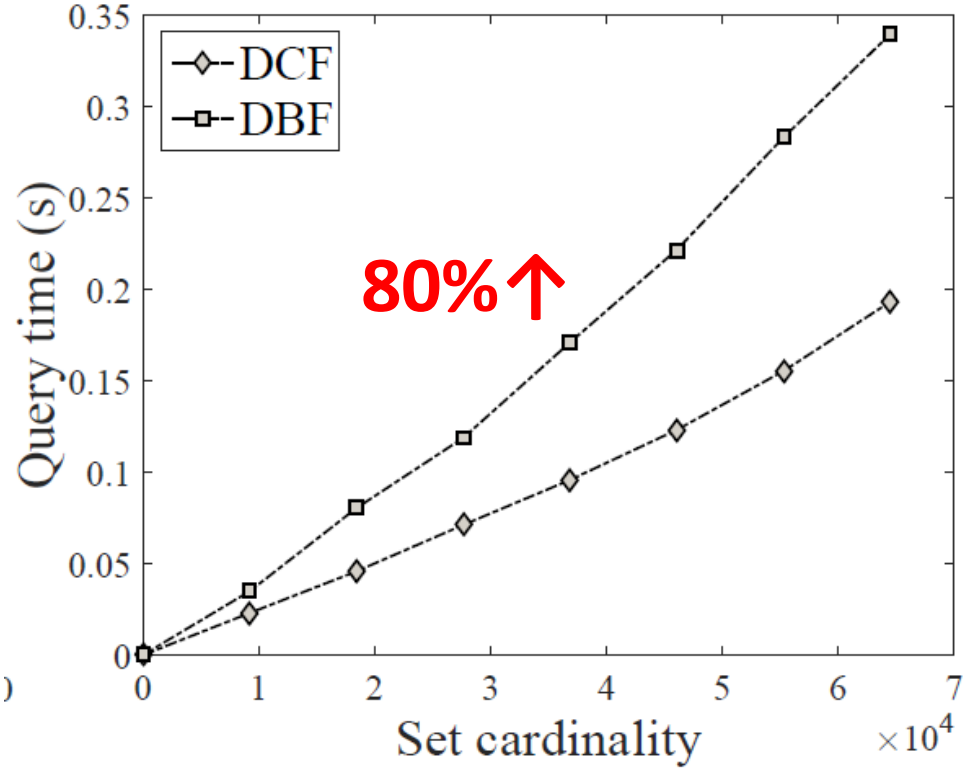


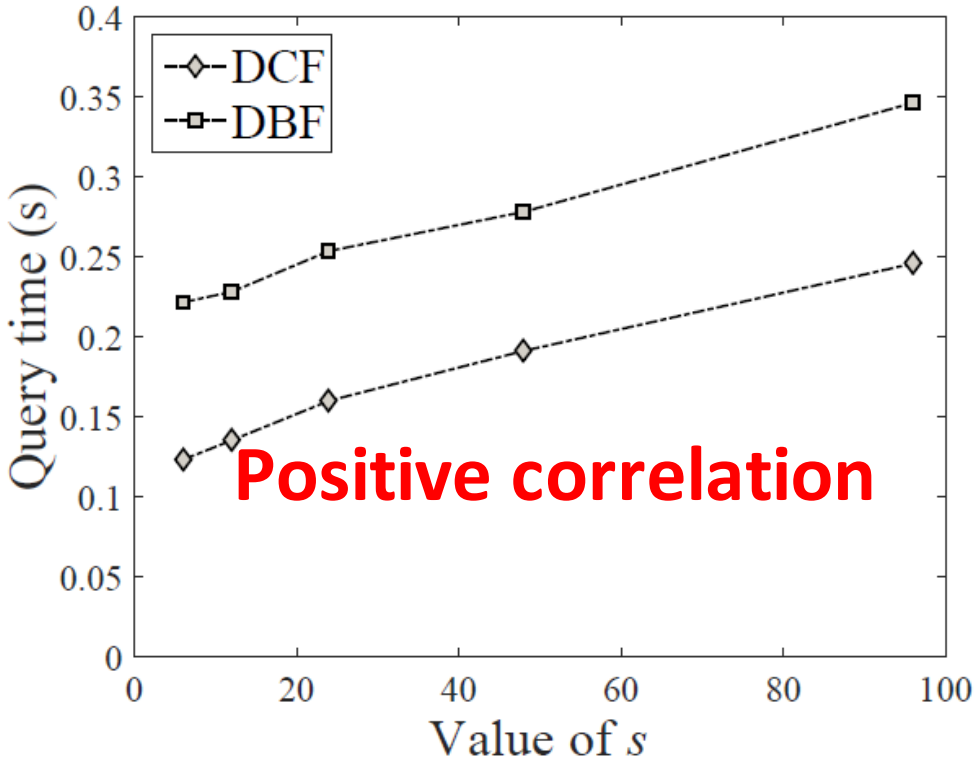
Insert Time


DCF
DBF
50%↑
Insert time (s)
Set cardinality
×10^4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1
2
3
4
5
6
7

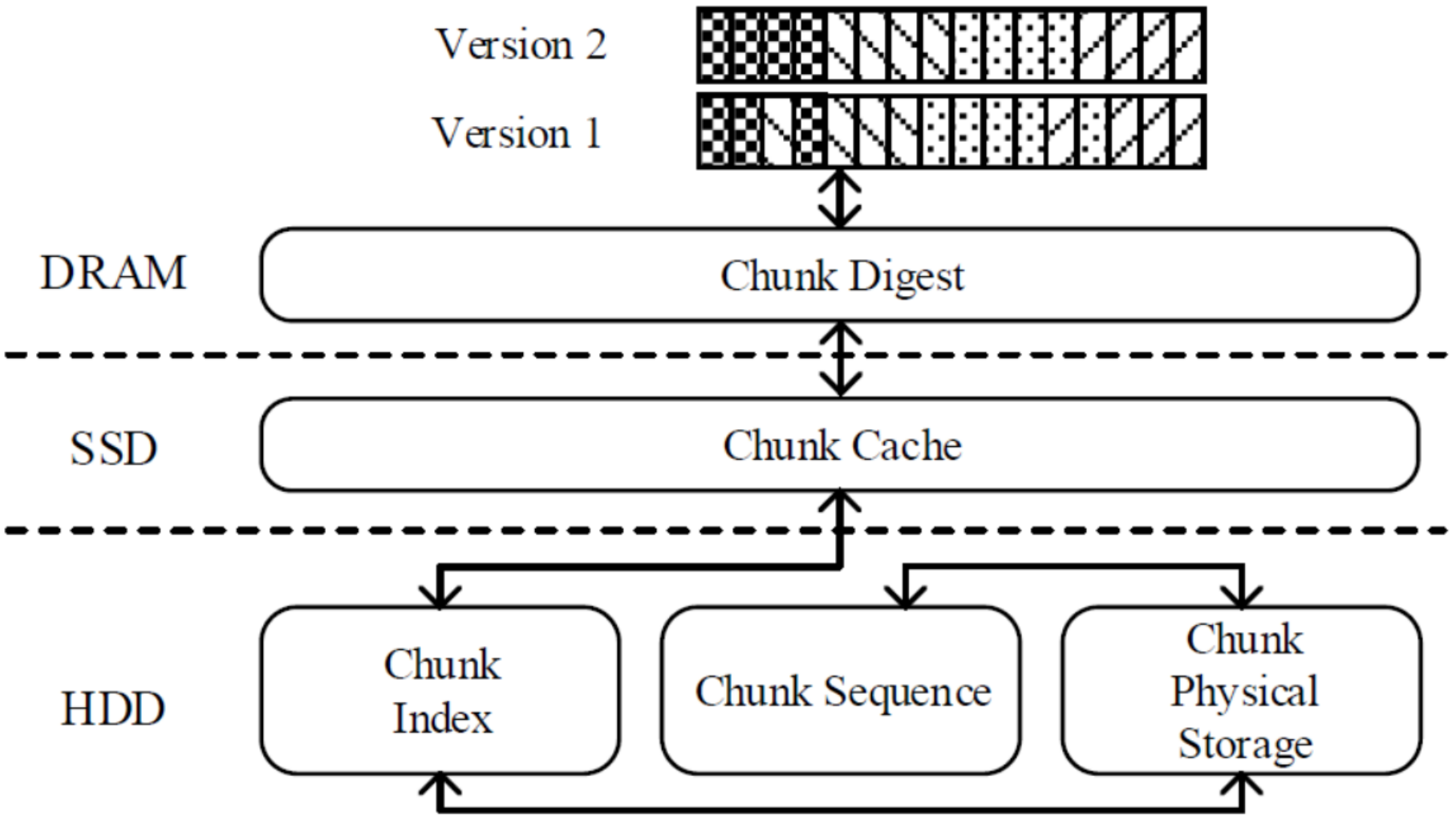

DCF
DBF
Changes slightly
Insert time (s)
Value of s
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
20
40
60
80
100

Query Time


DCF
DBF
80%↑
Query time (s)
Set cardinality
×10^4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0 1 2 3 4 5 6 7


DCF
DBF
Positive correlation
Query time (s)
Value of s
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0 20 40 60 80 100

Application in File Backup System

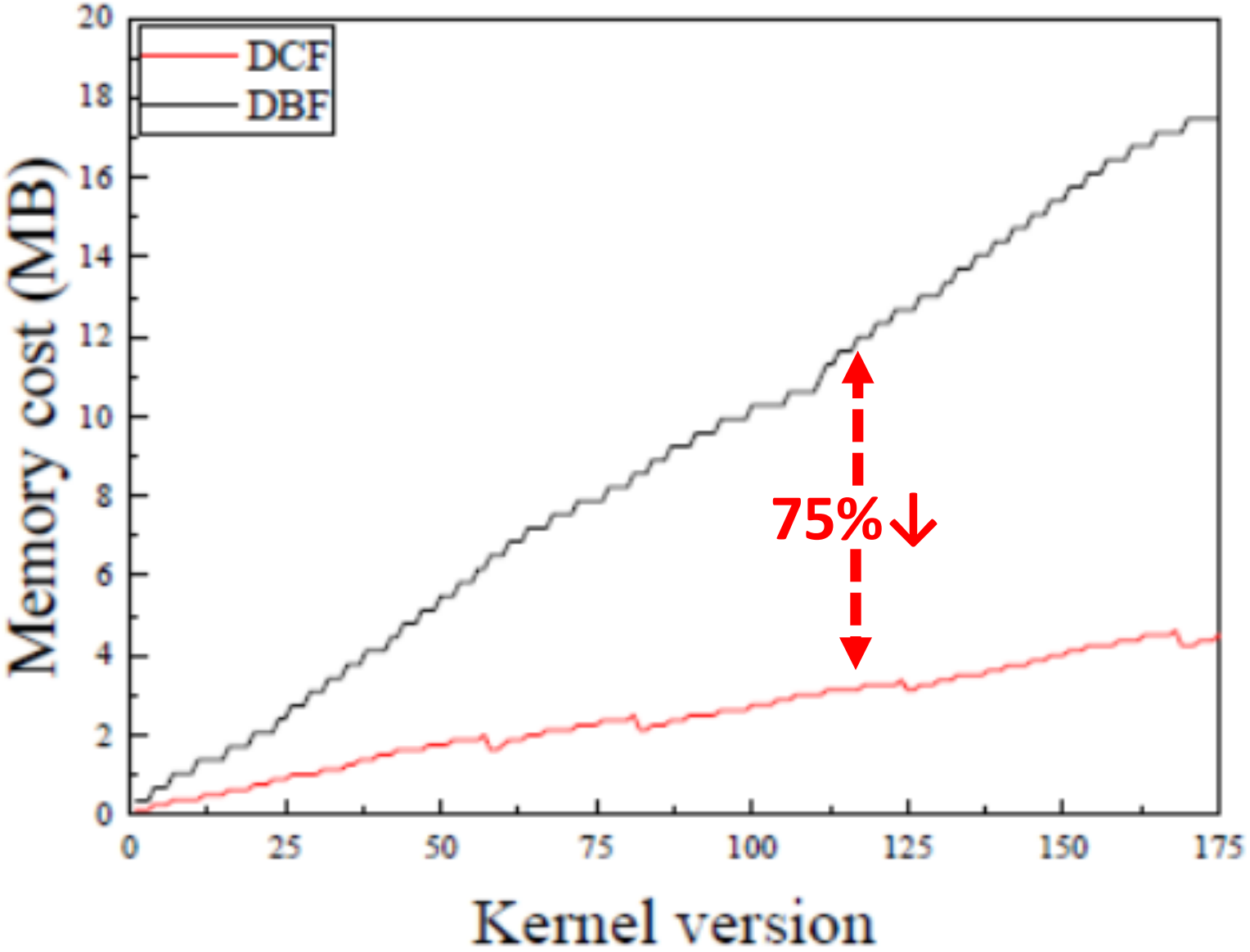


Data Sets

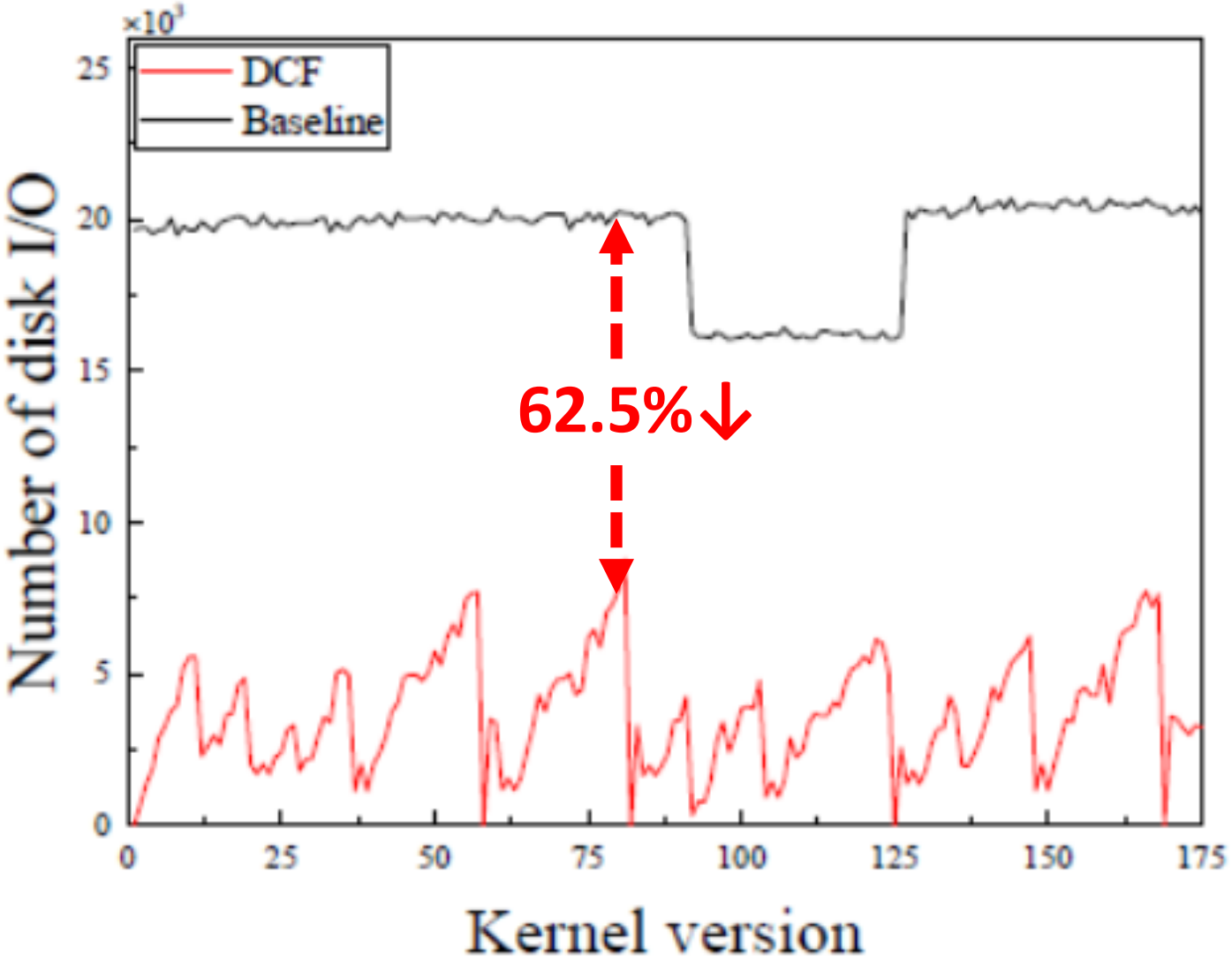
Dataset	Maximum set cardinality	False positive rate	# of building blocks	Versions
Linux Kernel Dataset	2,371,618	1.17×10^{-2}	1~50	175

https://www.kernel.org/pub/linux/kernel/, 2017.

Memory Cost Reduction



Disk I/O Reduction



Conclusion

- A novel DCF design for approximate set representation and membership testing for a dynamic set
- Support reliable element deletion and flexible expending
- Memory reduced by 75% compared to DBF

Thank you!

Download DCF Toolkit

https://github.com/CGCL-codes/DCF