Read as Needed: Building WiSER, a Flash-Optimized Search Engine

FAST '20

Jun He, Kan Wu, Sudarsun Kannan ,Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

Department of Computer Sciences, University of Wisconsin–Madison

Department of Computer Science, Rutgers University

Presented by Zhexin Jin and Yuming Xu, USTC, ADSL

April 24th, 2020

Outline

- Motivation
- Background
- Design
- Implementation
- Evaluation
- Conclusion

Outline

USTC, CHINA ADSLAB

- **Motivation**
- Background
- Design
- Implementation
- Evaluation
- Conclusion

Motivation

SSDs provide

- High throughput
- Low latency
- High read bandwidth
- Inexpensive

Many applications/systems have been optimized for SSDs

- Key-value stores: RocksDb, Wisckey, ...
- Graph stores: FlashGraph, Mosaic, ...
- File systems: SFS, F2FS, ...

But search engines are overlooked!

Motivation

Search engines require

- Low data latency: queries are interactive
- High data throughput: engines retrieve info from a large amount of data
- High scalability: data grows over time

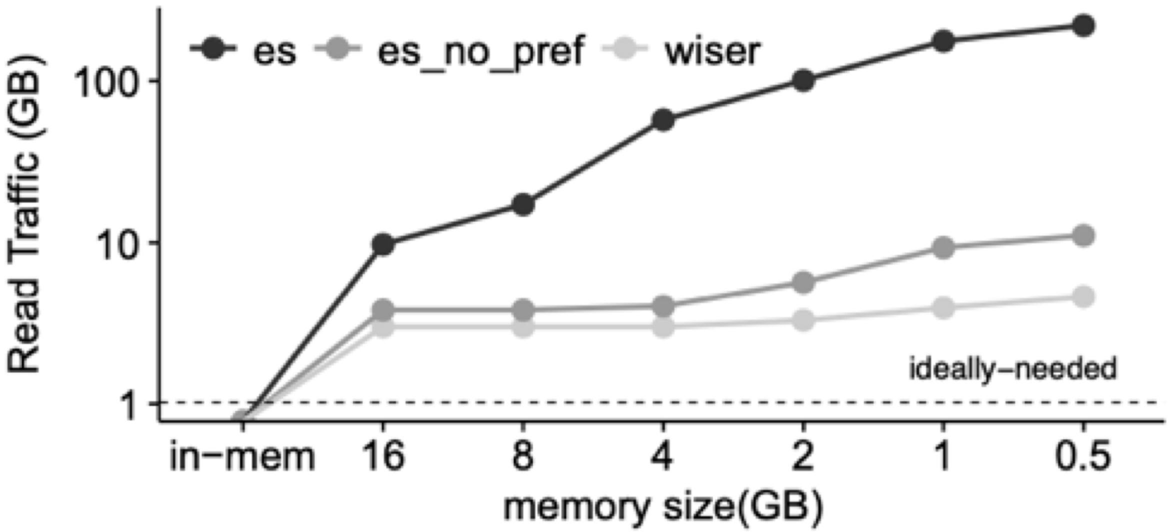
Just use more RAM?

- Cost prohibitive at large scale
- Data grows fast
- may waste bandwidth: rarely read and process 100GB/s

Motivation

USTC, CHINA ADSLAB

Can search engines perform well with a small memory and a fast SSD?



Outline

USTC, CHINA ADSLAB

- Motivation
- **Background**
- Design
- Implementation
- Evaluation
- Conclusion

Inverted index

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	I, thought, about, naming, the, engine, CHEESE, but, I, could, not, explain, CHEE
2	Fried, cheese, curds, cheddar, cheese, sale
3	Tofu, also, known, as, bean, curd, may, not, pair, well, with, cheese.

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	i, think, about, name, the, engine, cheese, but, i, can, not, explain, chee
2	fried, cheese, curd, cheddar, cheese, sale
3	tofu, also, know, as, bean, curd, may, not, pair, well, with, cheese

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	i, think, about, name, the, engine, cheese, but, i, can, not, explain, chee
2	fried, cheese, curd, cheddar, cheese, sale
3	tofu, also, know, as, bean, curd, may, not, pair, well, with, cheese

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

USTC, CHINA ADSLAB

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	i, think, about, name, the, engine, cheese, but, i, can, not, explain, chee
2	fried, cheese, curd, cheddar, cheese, sale
3	tofu, also, know, as, bean, curd, may, not, pair, well, with, cheese

TF: Term frequency

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

USTC, CHINA ADSLAB

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	i, think, about, name, the, engine, cheese, but, i, can, not, explain, chee
2	fried, cheese, curd, cheddar, cheese, sale
3	tofu, also, know, as, bean, curd, may, not, pair, well, with, cheese

POS: Position

Term Map

…
cheese
…
curd
…

Postings Lists

cheese →

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

curd →

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Inverted index

ID	Text
1	I thought about naming the engine CHEESE, but I could not explain CHEE.
2	Fried cheese curds, cheddar cheese sale.
3	Tofu, also known as bean curd, may not pair well with cheese.

ID	Tokens
1	i, think, about, name, the, engine, cheese, but, i, can, not, explain, chee
2	fried, cheese, curd, cheddar, cheese, sale
3	tofu, also, know, as, bean, curd, may, not, pair, well, with, cheese

OFF: Byte offset

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

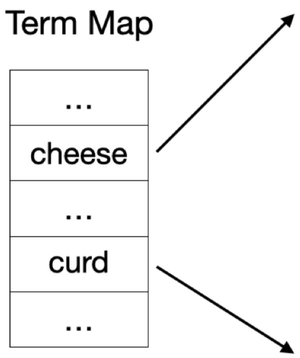
ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. The indexer splits a document into tokens.
2. The indexer transforms the tokens.
3. The location information of the term is inserted to a list, called a postings list.

Query processing

USTC, CHINA ADSLAB

Single-term query: cheese



Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

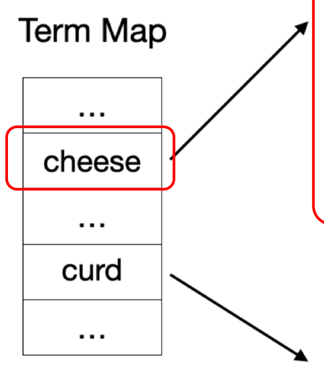
......

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

Single-term query: cheese



Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

......

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. Document matching: iterating document IDs in a term’s postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

Single-term query: cheese

Term Map

...
cheese
...
curd
...

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

......

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

USTC, CHINA ADSLAB

Single-term query: cheese

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF	Score
1	1	7	(34, 39)	xx
2	2	2, 5	(6, 11), (28, 33)	xx
3	1	12	(52, 57)	xx

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

USTC, CHINA ADSLAB

Single-term query: cheese

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF	Score
1	1	7	(34, 39)	xx
2	2	2, 5	(6, 11), (28, 33)	highest
3	1	12	(52, 57)	xx

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

Fried cheese curds, cheddar cheese sale.

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

Single-term query: cheese

Two-term query:

- cheese AND curd
- cheese OR curd

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

Query processing

USTC, CHINA ADSLAB

Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

Single-term query: cheese

Two-term query:

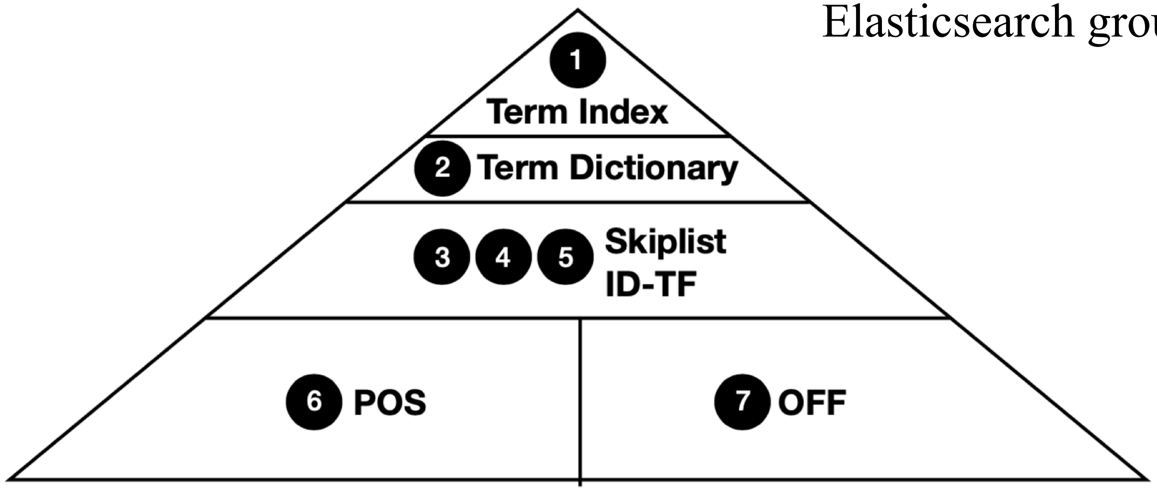
- cheese AND curd
- cheese OR curd

Phrase query:

- cheese curd

1. Document matching: iterating document IDs in a term's postings list.
2. Phrase matching: use positions to perform phrase matching.
3. Ranking: calculating the relevance score of each document, which usually uses TF.
4. Highlighting: highlighting queried terms in the top documents.

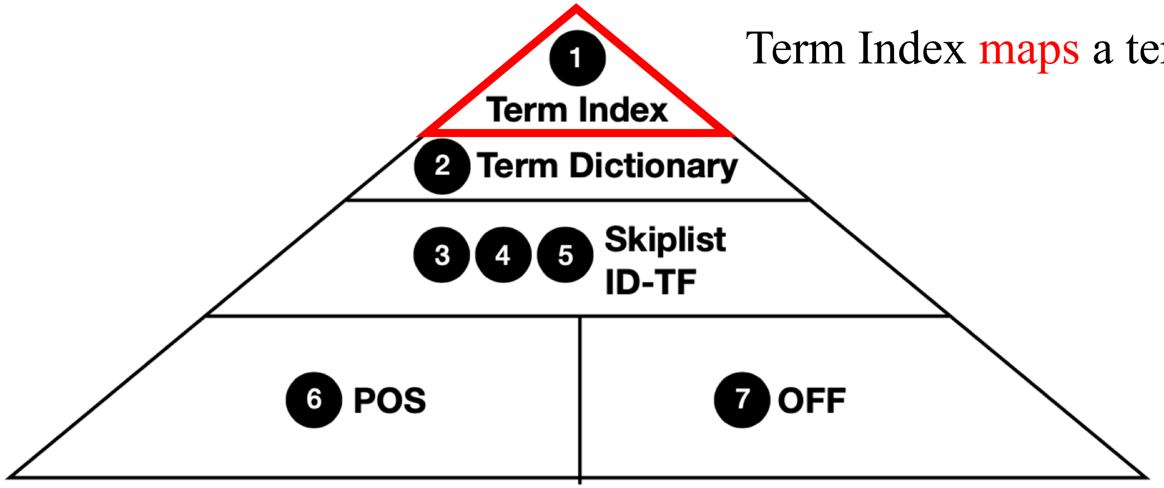
Elasticsearch



Elasticsearch groups data of different stages into multiple locations.

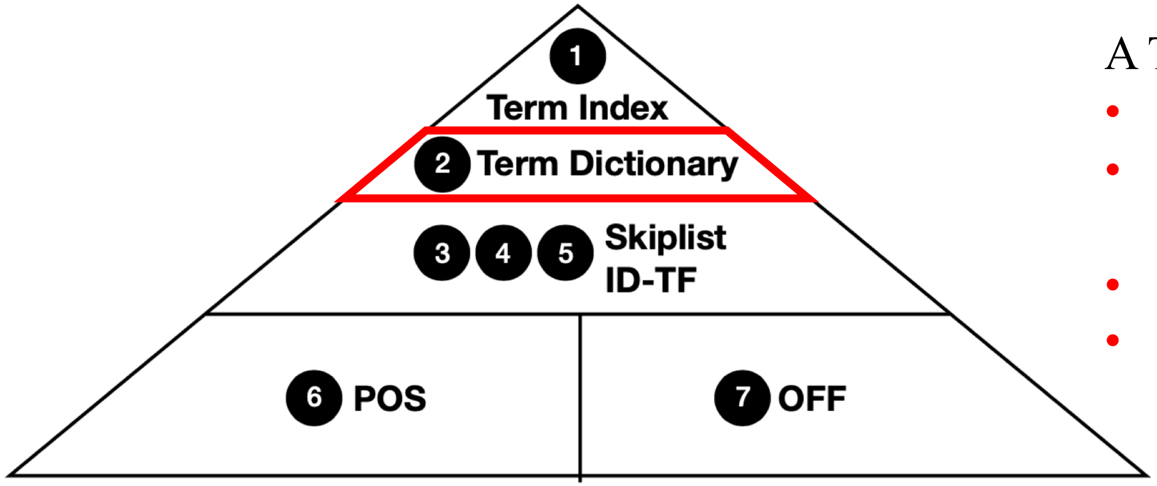
For Wikipedia
- Term Index: 4 MB
- Term Dictionary: 200 MB
- Skiplist, ID-TF: 2.7 GB
- POS: 4.8 GB
- OFF: 2.8 GB

Elasticsearch



Term Index maps a term to an entry in Term Dictionary.

Elasticsearch



A Term Dictionary entry contains

- metadata about a term (e.g., doc frequency)
- pointer pointing to document IDs and Term Frequencies (ID-TF)
- pointer pointing to positions (POS)
- pointer pointing to byte offsets (OFF).

Outline

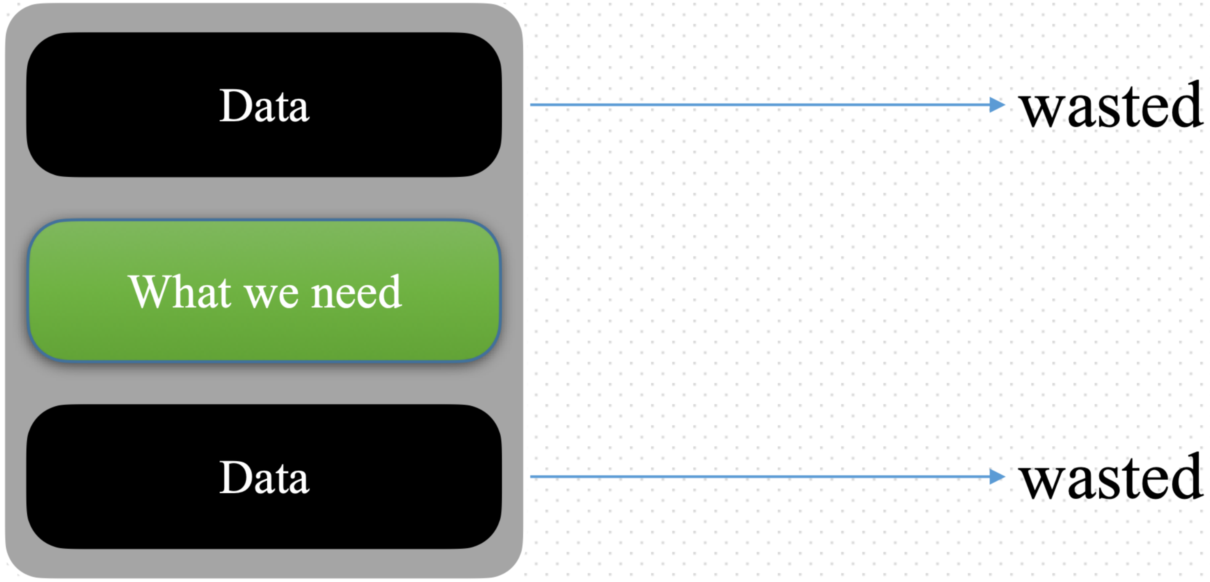
USTC, CHINA ADSLAB

- Motivation
- Background
- **Design**
- Implementation
- Evaluation
- Conclusion

Design

- **Read amplification**

In a word, read what we don’t need



File

Actually we fetch the whole file from disk

Design

USTC, CHINA ADSLAB

Four techniques to reach our goals

Cross-stage data grouping

- reduce read amplification
- make I/O requests be large

Two-way Cost-aware Bloom Filter

- also reduce read amplification

Adaptive prefetching

- hide I/O latency

Trade Disk Space for I/O

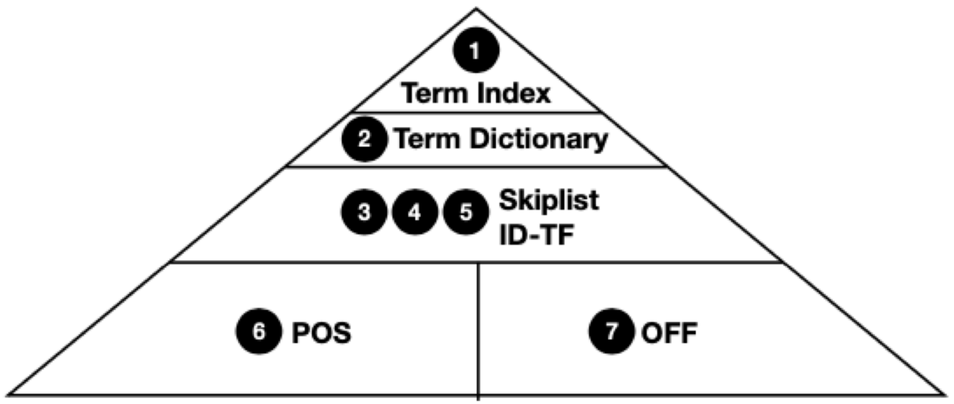
- reduce read amplification

Cross-stage data grouping

USTC, CHINA ADSLAB

In the Background part, we have known the process of a query

In single term query or phrase query, we all need to read 1-6



Term Map

…
cheese
…
curd
…

Postings Lists

ID	TF	POS	OFF
1	1	7	(34, 39)
2	2	2, 5	(6, 11), (28, 33)
3	1	12	(52, 57)

……

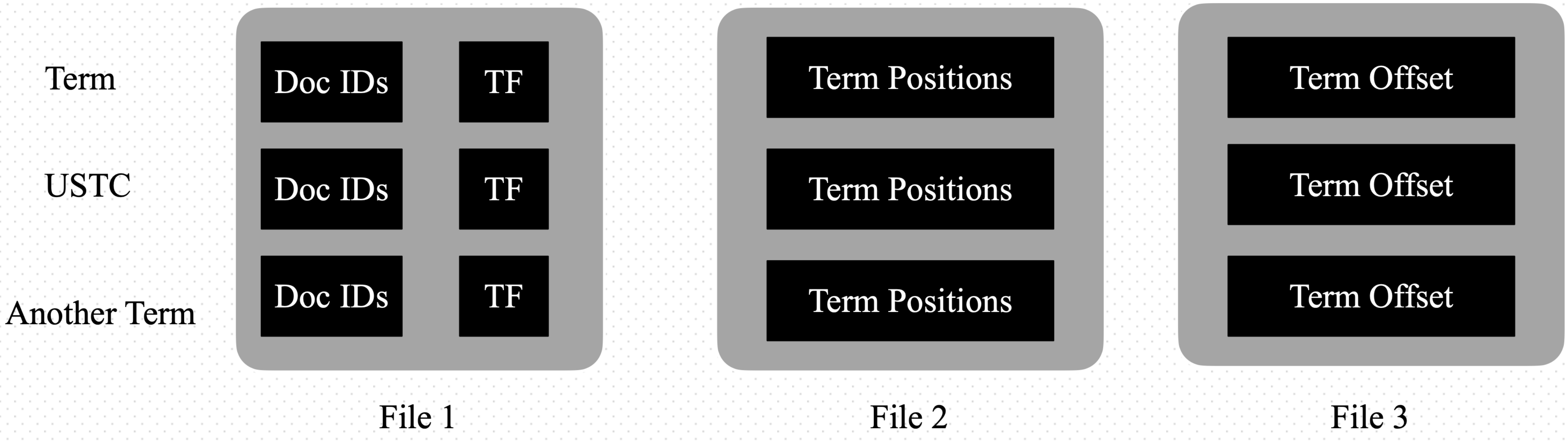
ID	TF	POS	OFF
2	1	3	(13, 17)
3	1	6	(25, 28)

Cross-stage data grouping

USTC, CHINA ADSLAB

In the previous design, we need to read disk many times

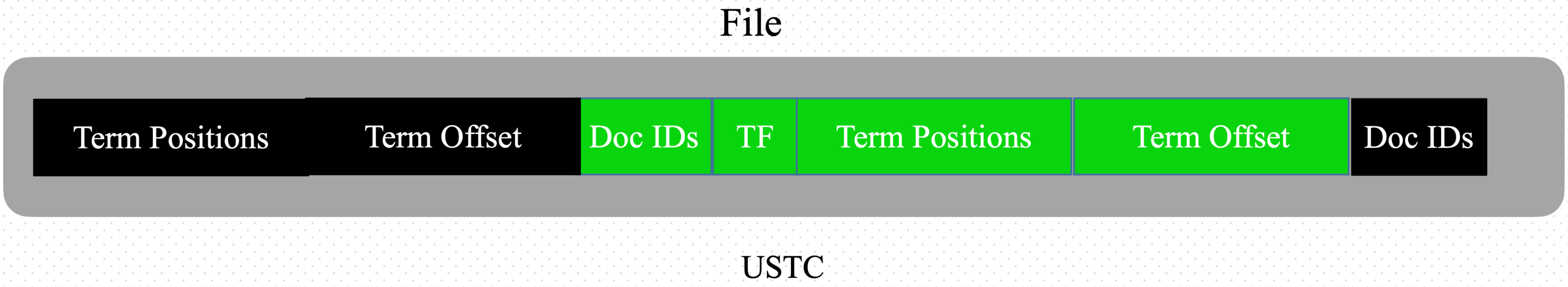
For term ‘USTC’, IO count:3 or more (TF:Term Frequencies)



Cross-stage data grouping

WISER change the grouping way

For term ‘USTC’, IO count:1

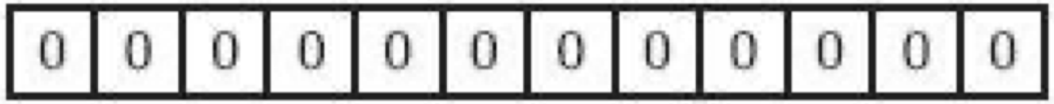


Two-way Cost-aware Bloom Filter

Bloom Filter

- **A type of Data Structure**
- **Use hash to test whether a element is in the set**

At first, Bloom Filter is a bit-array containing m bits,
and all bits are set to 0

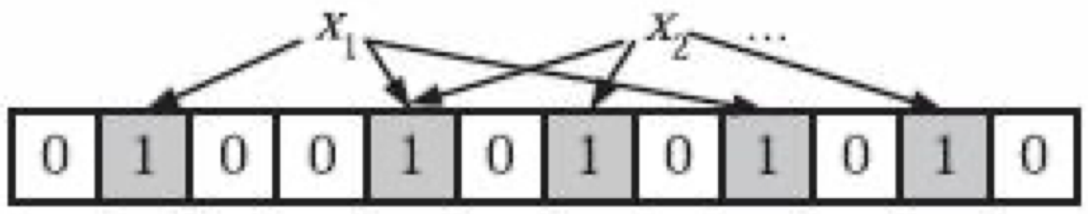


Two-way Cost-aware Bloom Filter

Bloom Filter

- **A type of Data Structure**
- **Use hash to test whether a element is in the set**

S = {x1,x2,…,xn}
Using k individual hash functions to map the element



For example, if we want to present $x_1 \in BF$, $BF[h_i(x)], i = 1,...,k$ should be set to 1

Two-way Cost-aware Bloom Filter

Bloom Filter

- **A type of Data Structure**
- **Use hash to test whether a element is in the set**

S = {x1,x2,…,xn}
Using k individual hash functions to map the element

Check whether $y \in BF$:
check $BF[h_i(y)]$, $i = 1,\ldots,k$, if
$\forall i, BF[h_i(y)] = 1$, then $y \in BF$

Two-way Cost-aware Bloom Filter

Bloom Filter

- **For phrase query like ‘Distributed System’**
- **100% recall but precision<100%**
- **For each term in each document**
- **In this case Bloom Filter aim at optimizing negative result**

Two conditions: 1. the percentage of negative tests must be high

2. Reading Bloom Filter must be faster than directly reading position

Two-way Cost-aware Bloom Filter

Using plain Bloom Filter

- Two conditions are conflict!
- One way is slow while another way is fast
- May be larger than positions

Two-way Cost-aware Bloom Filter

USTC, CHINA ADSLAB

Using plain Bloom Filter

- **Two conditions are conflict!**
- **One way is slow while another way is fast**
- **May be larger than positions**

the percentage of negative tests must be high :
Bloom Filter should be large
Reading Bloom Filter must be faster than directly reading position :
Bloom Filter should be small

Two-way Cost-aware Bloom Filter

USTC, CHINA ADSLAB

Using plain Bloom Filter

- Two conditions are conflict!
- **One way is slow while another way is fast**
- May be larger than positions

60KB < 50KB + 500KB

Using two-way Bloom Filter is better

Distributed

Filter after:60KB

Positions:50KB

Systems

Filter Before:600KB

Positions:500KB

600KB > 50KB + 500KB

Two-way Cost-aware Bloom Filter

Using plain Bloom Filter

- Two conditions are conflict!
- One way is slow while another way is fast
- **May be larger than positions**

Two-way doesn't work, cost-aware is added

Distributed	Filter before:600KB	Filter after:600KB	Positions:200KB
Systems	Filter before:600KB	Filter after:600KB	Positions:200KB

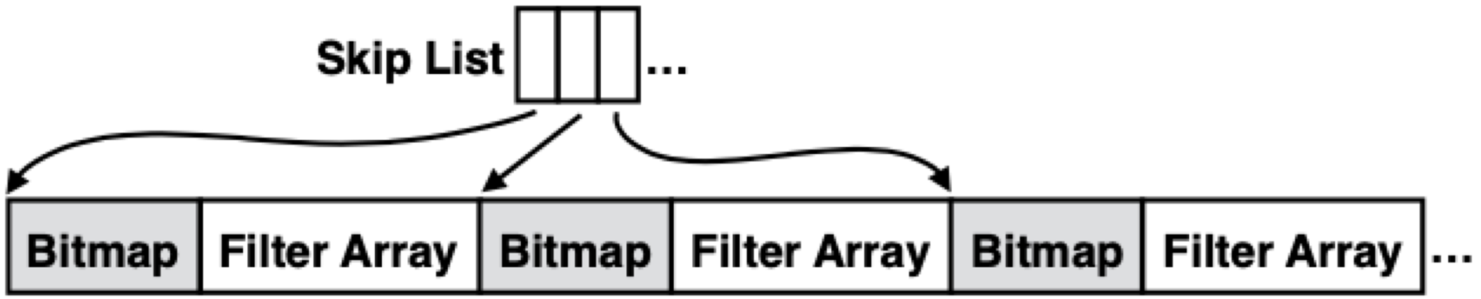
600KB > 200KB + 200KB

Two-way Cost-aware Bloom Filter

USTC, CHINA ADSLAB

- **Allocate Bloom Filter**

  To further reduce the size of BFs, using bitmap-based data layout to store BFs



  Using skip list to avoid reading large chunks of filters

  Using bitmap to reduce the space usage of empty BFs

Adaptive Prefetch

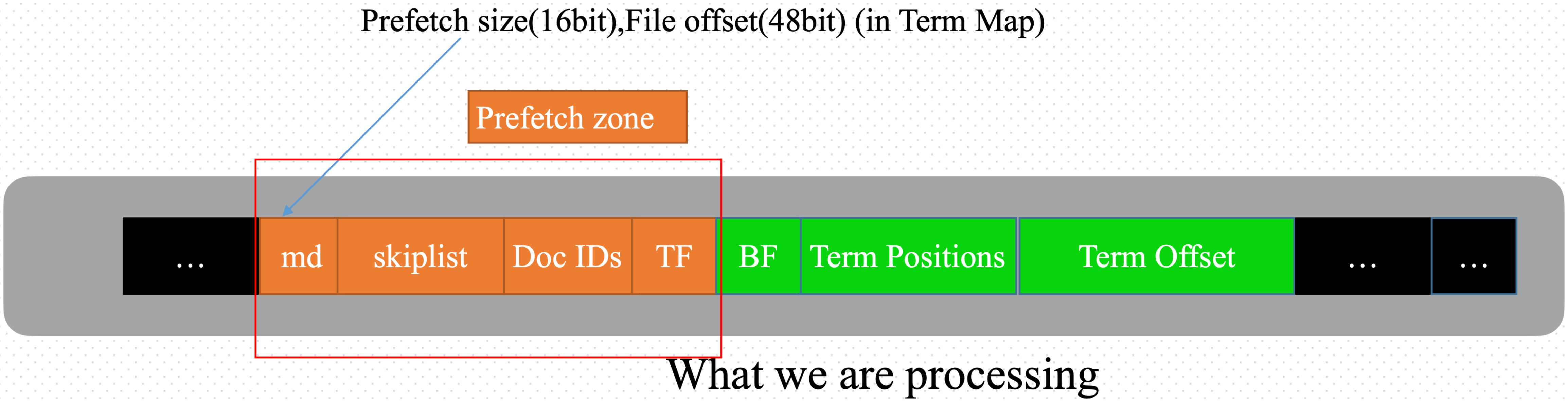
USTC, CHINA ADSLAB

- **Elasticsearch using native prefetch**
  - Linux unconditionally prefetches data of a fixed size(default:128KB)
  - But data sizes are different
  - Cause high read amplification

Adaptive Prefetch

USTC, CHINA ADSLAB

WISER adaptively prefetches frequently-used data to hide I/O latency

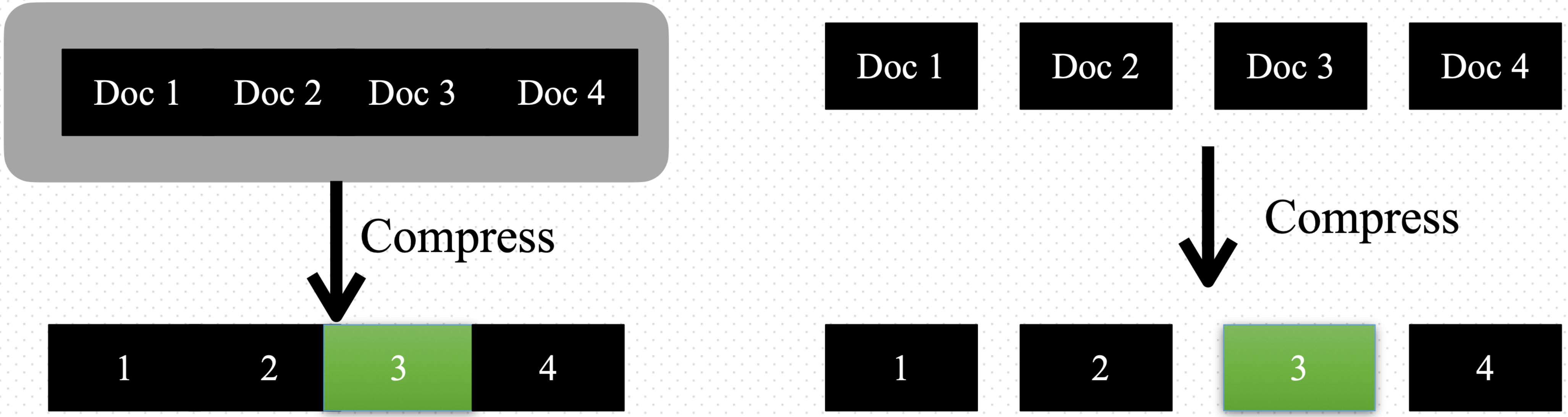


Adaptive: only when all prefetch zones in a query are larger than a threshold(e.g.,128KB), and divide prefetch zone to avoid access to much data at a time

Trade Disk Space for I/O

WISER compresses documents individually to reduce read amplification

4KB align(just as the size of one SSD page)



Need to fetch all data

Only to fetch 3

Impact on Indexing

- **Focus on optimizing query processing instead of index creation**

  query processing is performed far more frequently

- **Cross-stage data grouping**

  Does not add overhead, data is just placed in different place

- **Adaptive prefetching**

  Employs existing info, does not add any overhead

- **Trading space for I/O**

  Adds I/O overhead for indexing because document need more space

Impact on Indexing

USTC, CHINA ADSLAB

- **Bloom Filter**

  Requires extra computation: building BF

  Although many filters are empty, the accumulative cost can be high.

  Currently, haven't optimize the process of building

  todo: 1. Parallelize the building process

  2. cache the hash values of popular terms to avoid hashing the same term frequently

Outline

- Motivation
- Background
- Design
- **Implementation**
- Evaluation
- Conclusion

Implementation

USTC, CHINA ADSLAB

- **11000 lines of C++ code (es is based on Java)**

- Using mmap( ) to map data file
- Switch from class virtualization to templates
- Use case-specific functions to allow special optimizations
- Reusing preallocated std::vector to avoided frequent memory allocation

Outline

USTC, CHINA ADSLAB

- Motivation
- Background
- Design
- Implementation
- **Evaluation**
- Conclusion

Setup

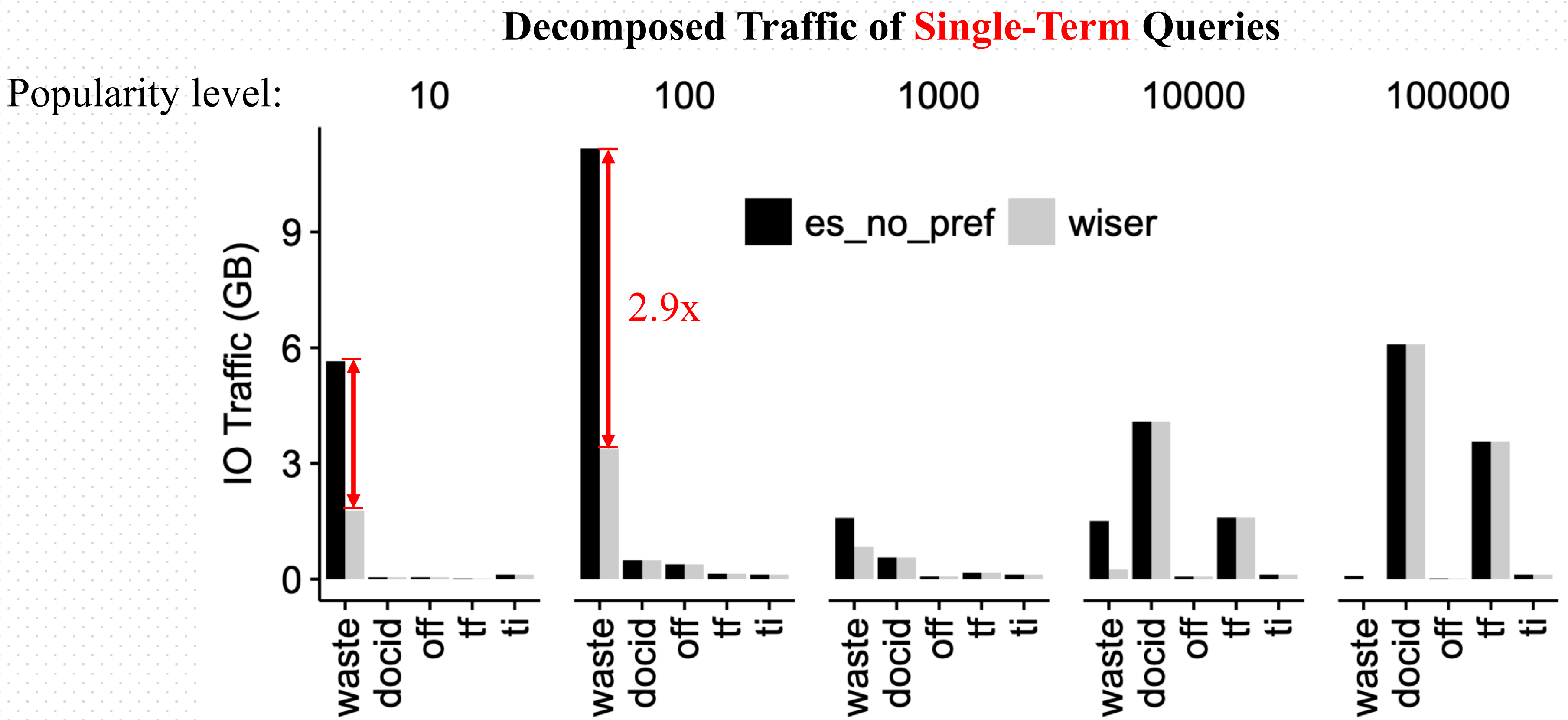
- 16 CPU cores
- 64-GB RAM
- 256-GB NVMe SSD
  - Peak read bandwidth is 2.0 GB/s
  - Peak IOPS is 200,000
- Ubuntu with Linux 4.4.0
- Use only 512 MB of memory (using a Linux container)

WSBench

- Dataset: Wikipedia
  - Total size: 18 GB
  - 6 million documents, 6 million unique terms
- Queries:
  - single term queries, “and” queries, “phrase” queries, real queries
  - vary term popularities in wikipedia

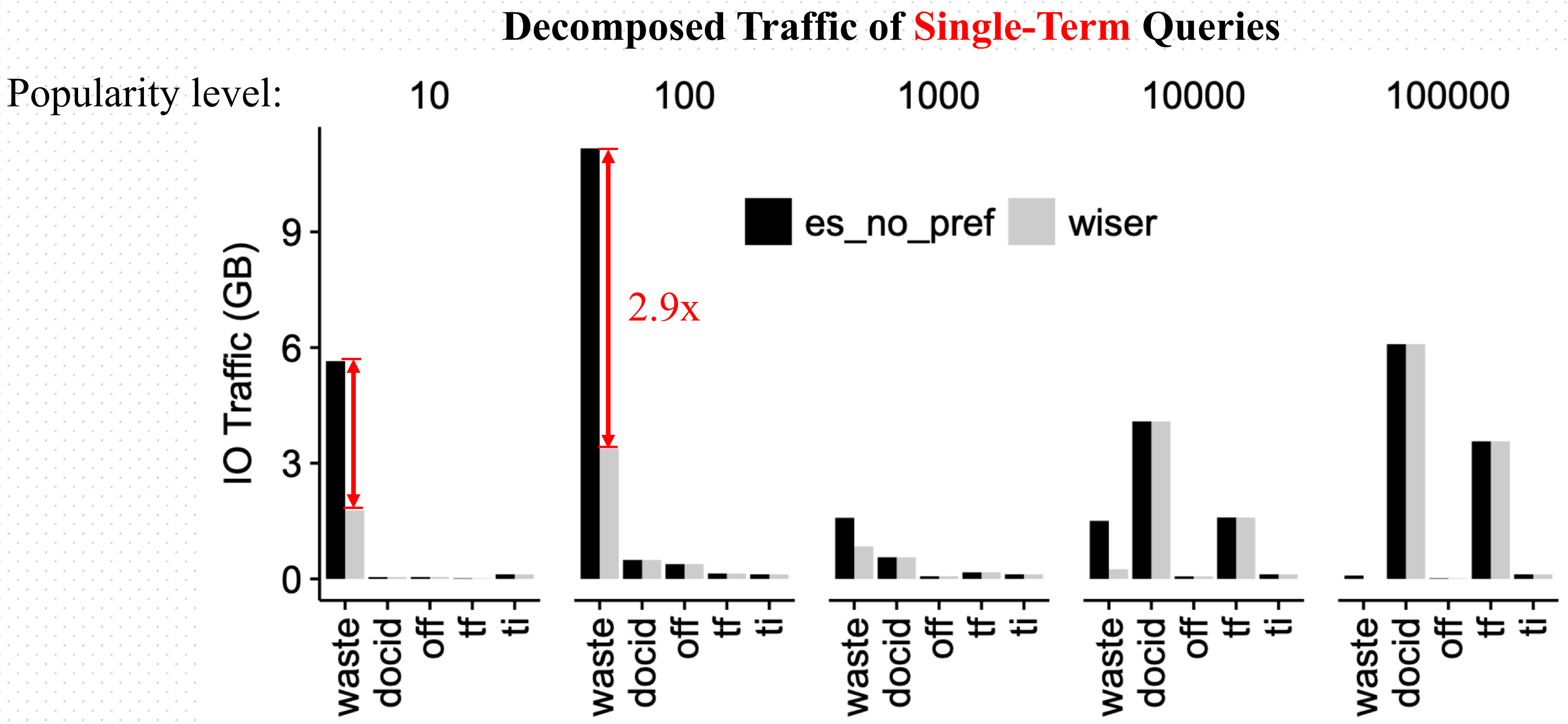
popularity level = document frequency = the number of documents in which a term appears

Cross-stage Data Grouping



- waste: the data that is unnecessarily read
- docid: the ideally needed data of document ID
- off: offset
- tf: term frequency
- ti: term index/dictionary
- es_no_pref: Elasticsearch without prefetch

Cross-stage Data Grouping



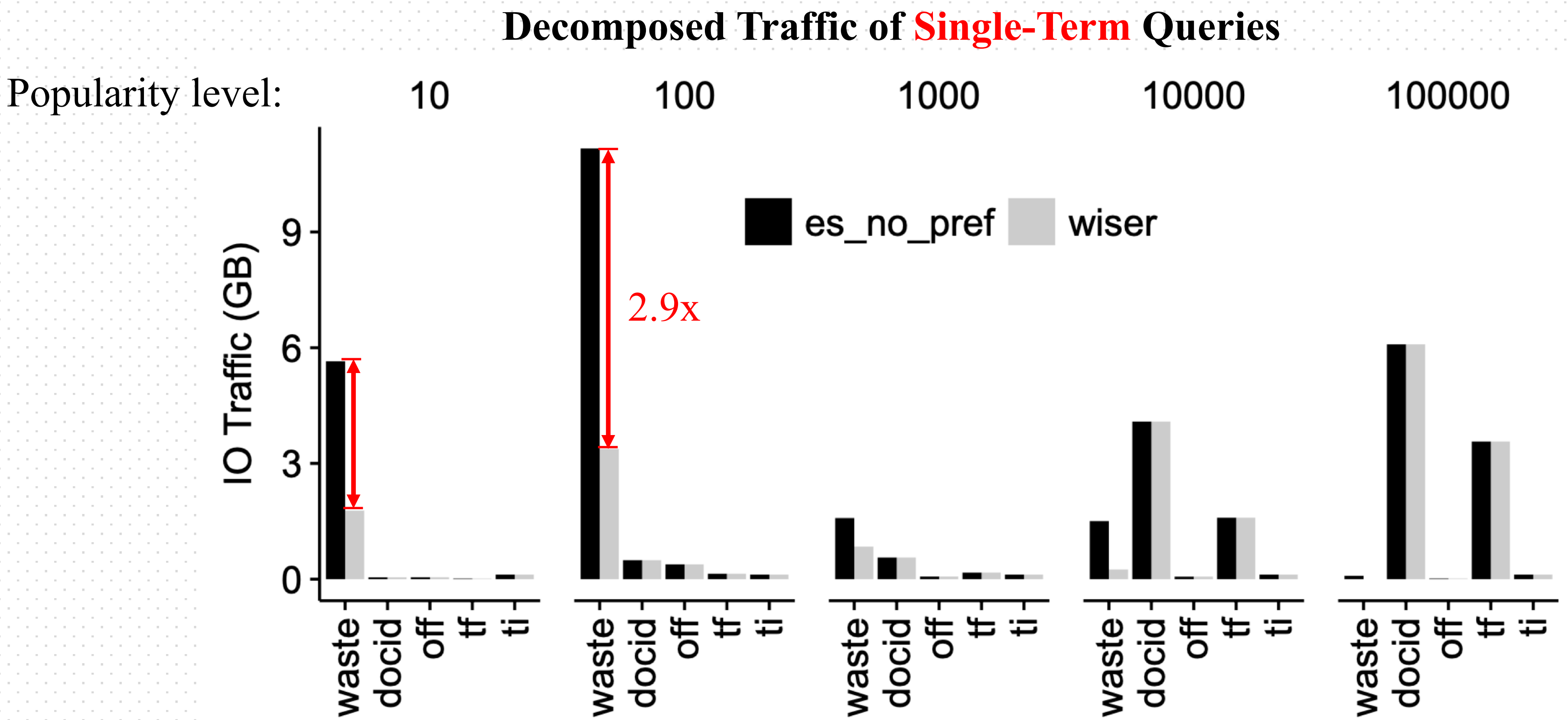
Elasticsearch needs three separate I/O requests:

1. Term index   2. ID, TF   3. Offsets

WiSER only needs one I/O request!

Cross-stage Data Grouping

USTC, CHINA ADSLAB

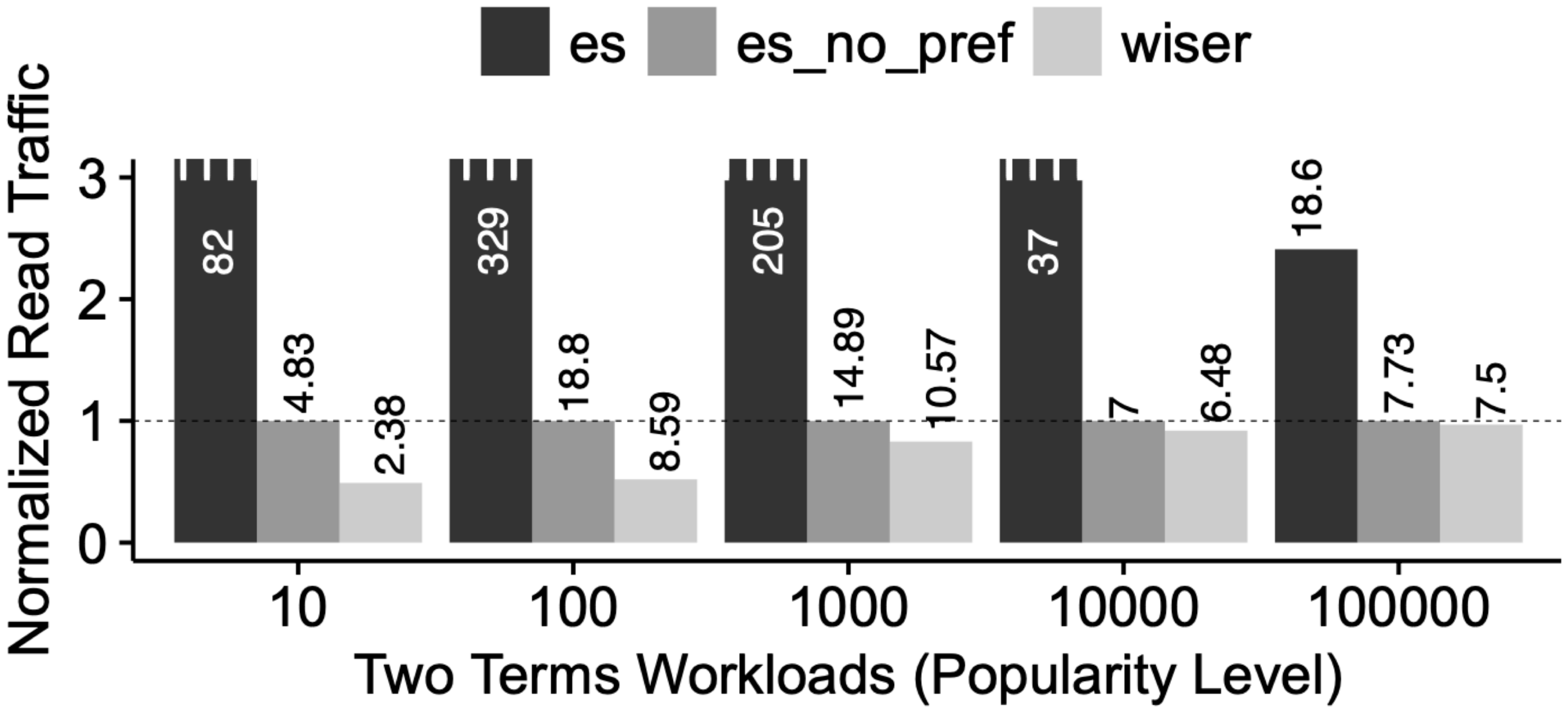


Elasticsearch needs three separate I/O requests:
1. Term index 2. ID, TF 3. Offsets
WiSER only needs one I/O request!

Reduce read amplification!

Cross-stage Data Grouping

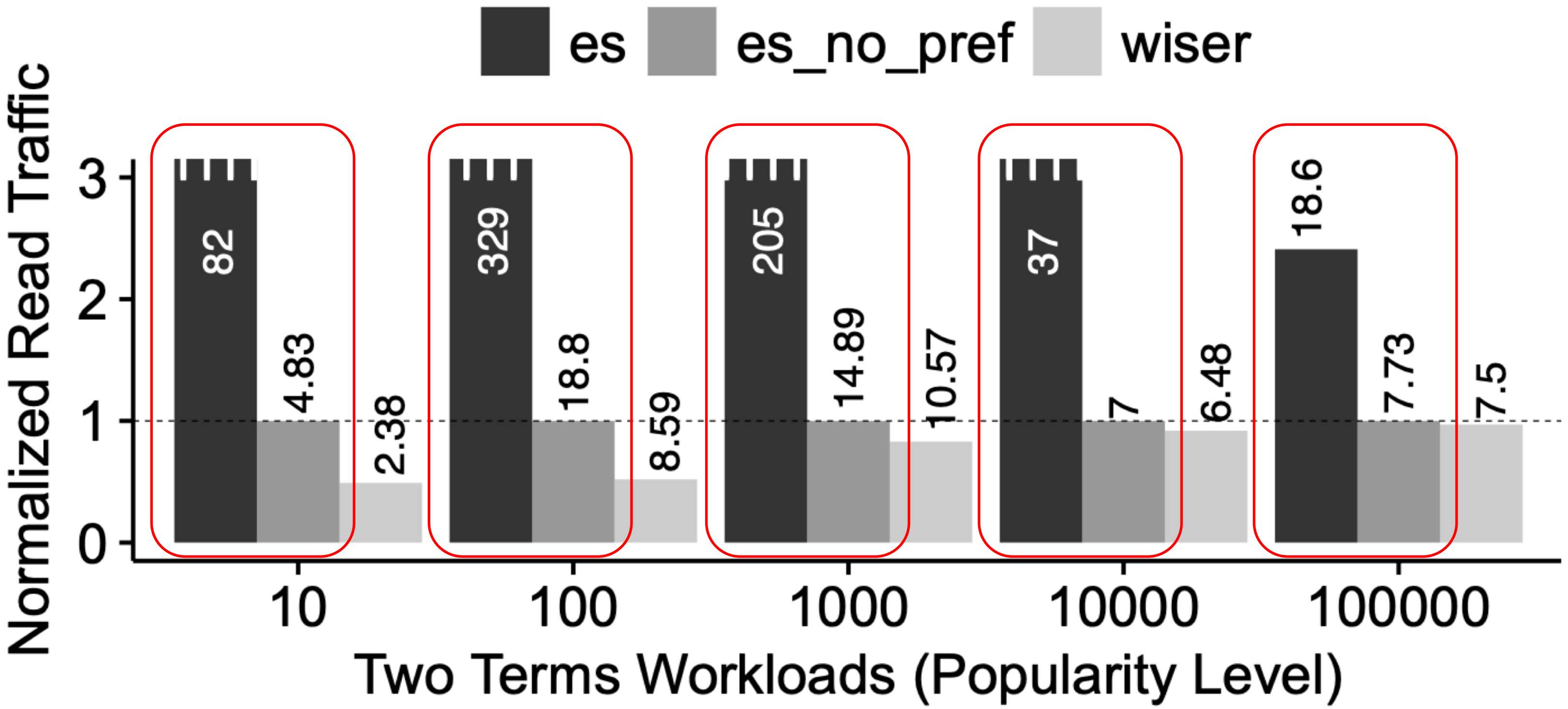
I/O Traffic of Two-term Match Queries



Cross-stage Data Grouping

USTC, CHINA ADSLAB

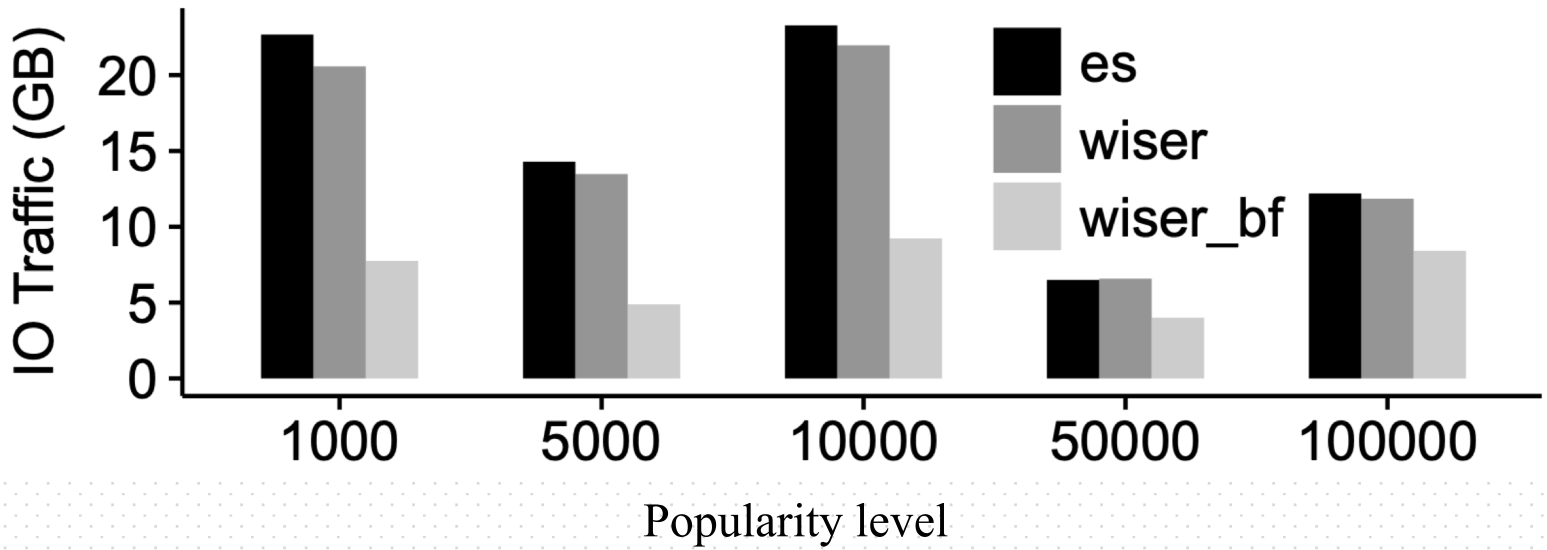
I/O Traffic of Two-term Match Queries



Naive prefetch in Elasticsearch can increase read amplification significantly!

Two-Way Cost-Aware Bloom Filters

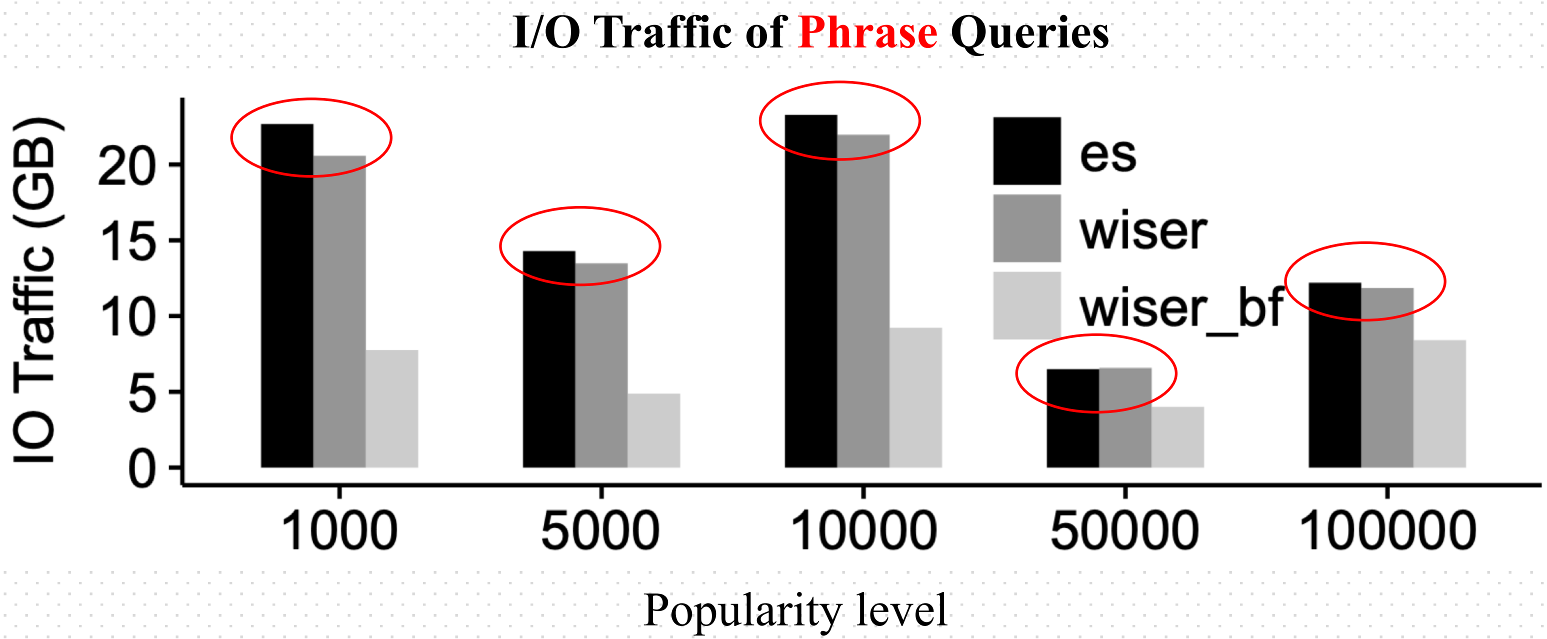
I/O Traffic of Phrase Queries



- es: Elasticsearch without prefetch
- wiser: WiSER without Bloom filters
- wiser_bf: WiSER with Bloom filters

Two-Way Cost-Aware Bloom Filters

USTC, CHINA ADSLAB



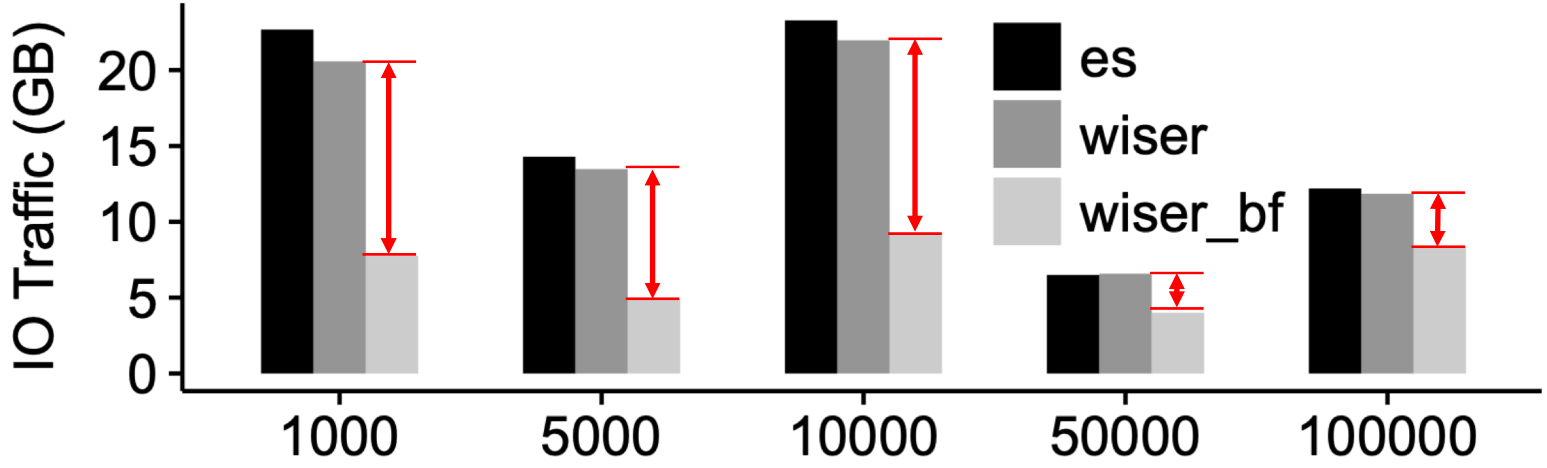
- es: Elasticsearch without prefetch
- wiser: WiSER without Bloom filters
- wiser_bf: WiSER with Bloom filters

WiSER without our Bloom filters demands a similar amount of data as Elasticsearch.

Two-Way Cost-Aware Bloom Filters

USTC, CHINA ADSLAB

I/O Traffic of Phrase Queries



Popularity level

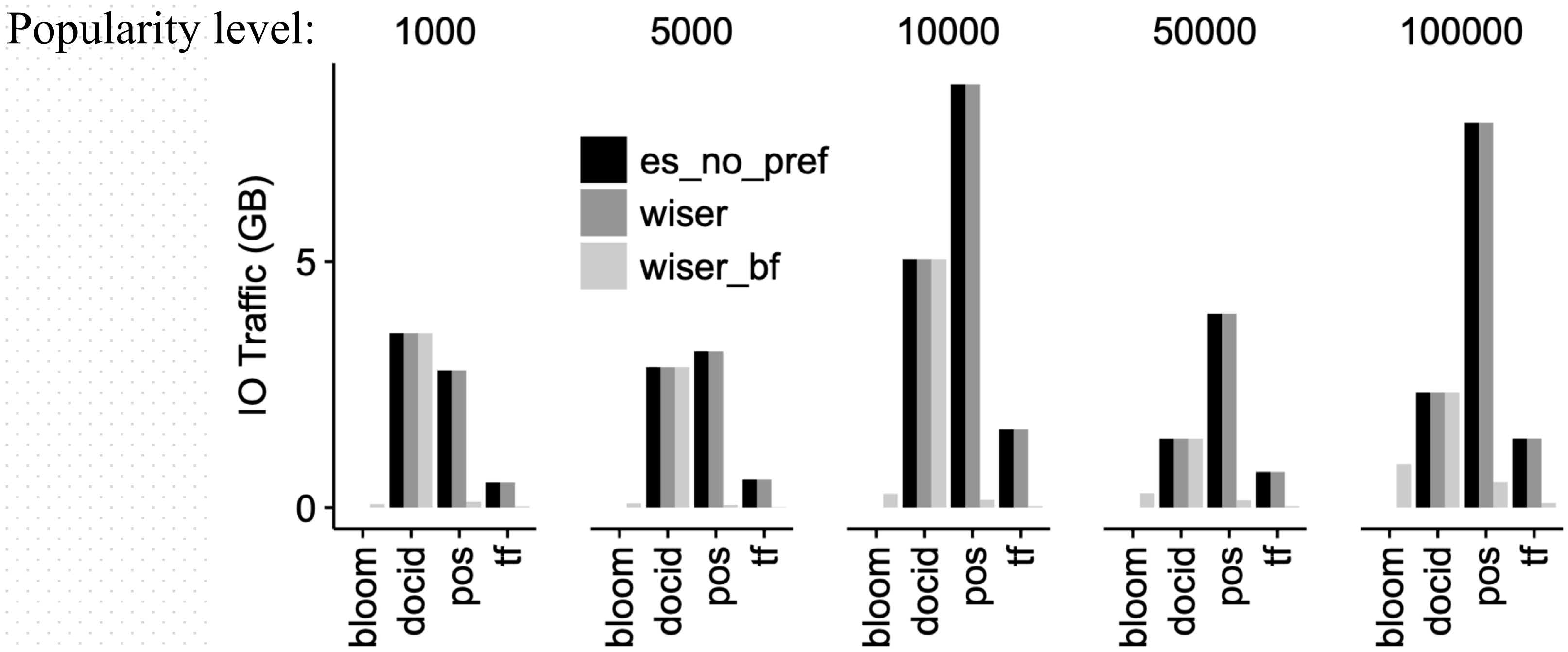
- es: Elasticsearch without prefetch
- wiser: WiSER without Bloom filters
- wiser_bf: WiSER with Bloom filters

WiSER with Bloom filters incurs much less I/O traffic!

Two-Way Cost-Aware Bloom Filters

USTC, CHINA ADSLAB

Decomposed Traffic Analysis of Phrase Queries



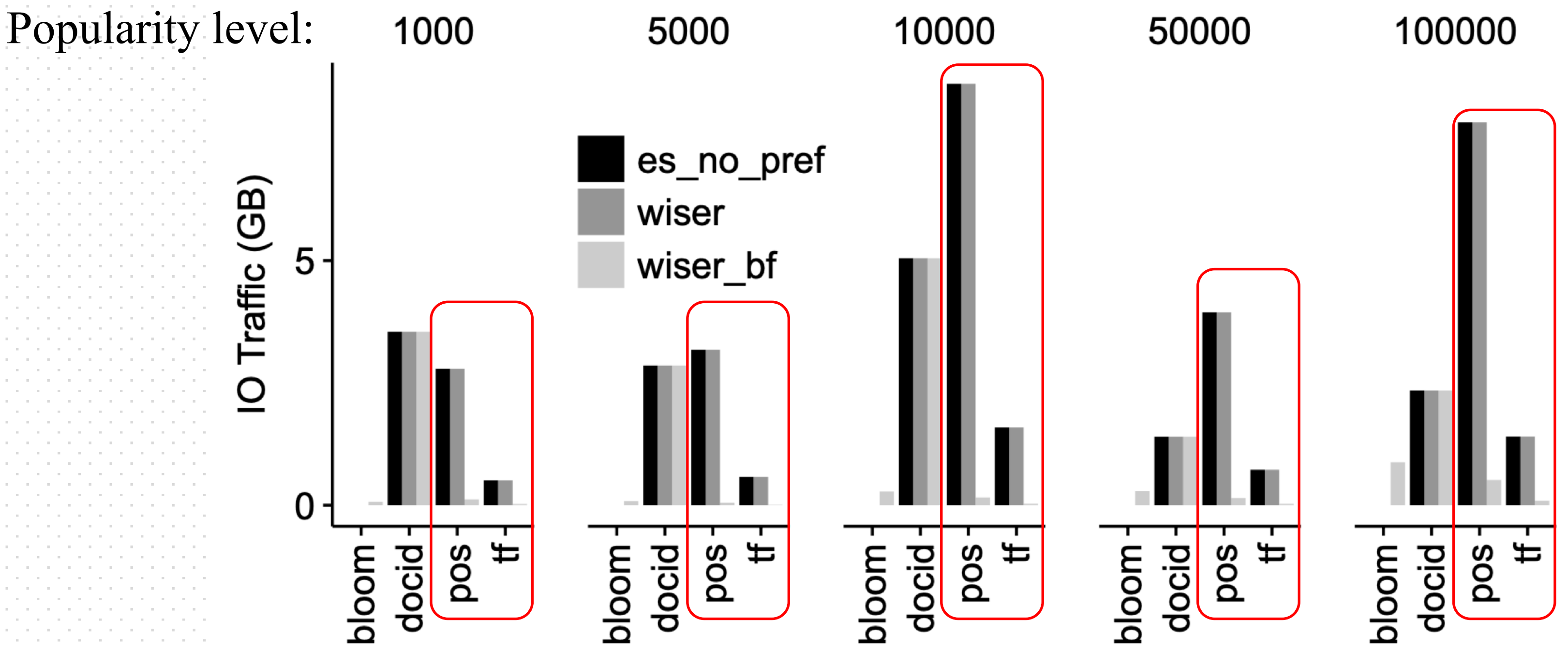
- bloom: the ideally needed data of Bloom filters
- docid: document ID
- pos: positions
- tf: term frequencies

- es_no_pref: Elasticsearch without prefetch
- wiser: WiSER without Bloom filters
- wiser_bf: WiSER with Bloom filters
- ideally: byte-addressable

Two-Way Cost-Aware Bloom Filters

USTC, CHINA ADSLAB

Decomposed Traffic Analysis of Phrase Queries



Reduce the traffic from positions and term frequencies!

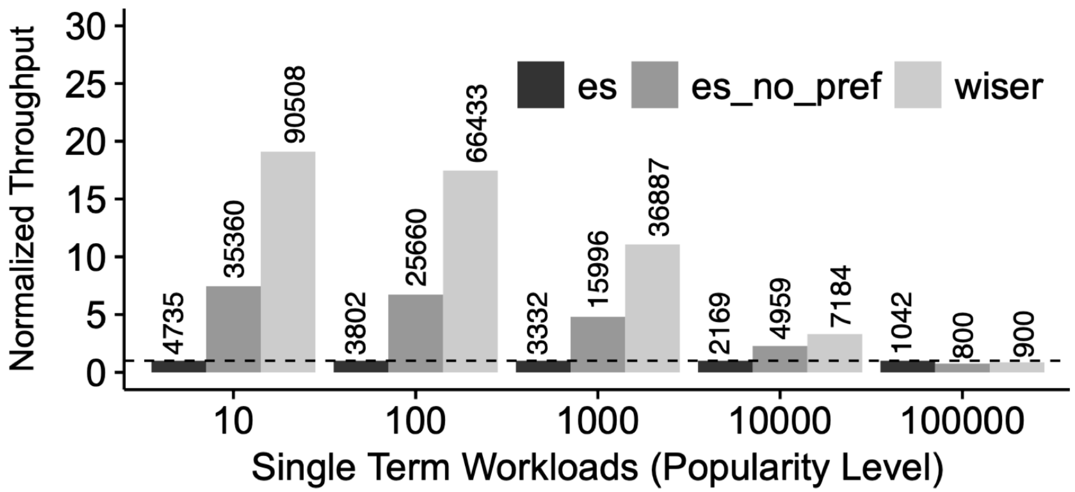
Adaptive Prefetching

Trade Disk Space for Less I/O

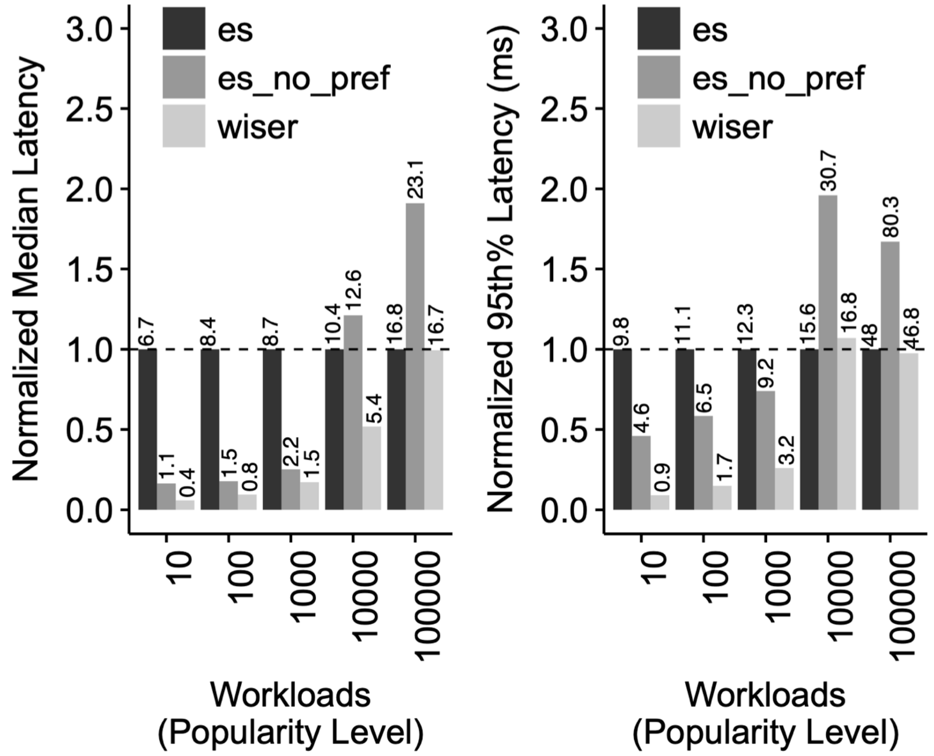
End-to-end Performance

USTC, CHINA ADSLAB

Single Term Matching Throughput



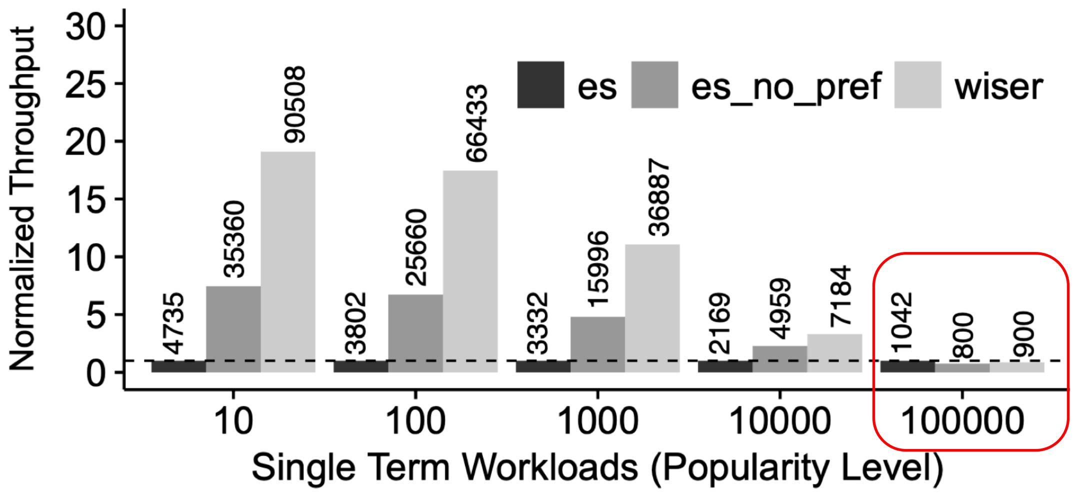
Single Term Matching Latency



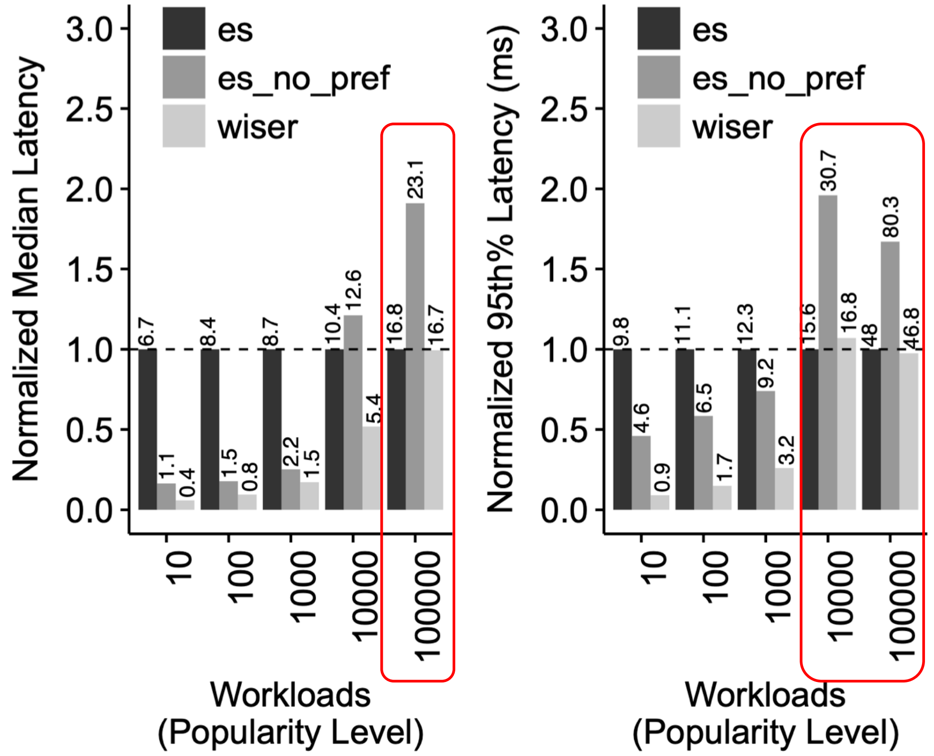
- Throughput: QPS (Queries Per Second)
- es: Elasticsearch with prefetch (128 KB)
- es_no_pref: Elasticsearch without prefetch
- wiser: WiSER

End-to-end Performance

Single Term Matching Throughput



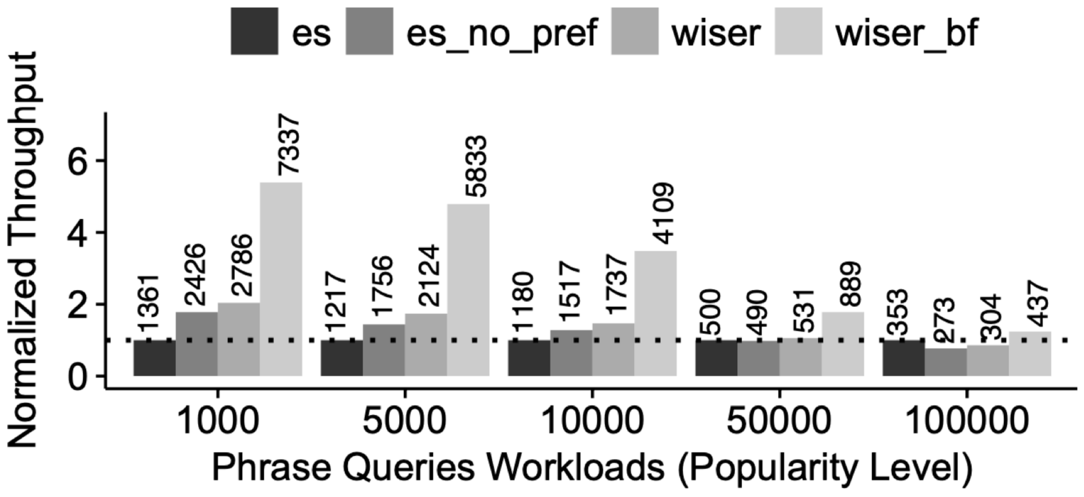
Single Term Matching Latency



Due to WiSER's less efficient score calculation.

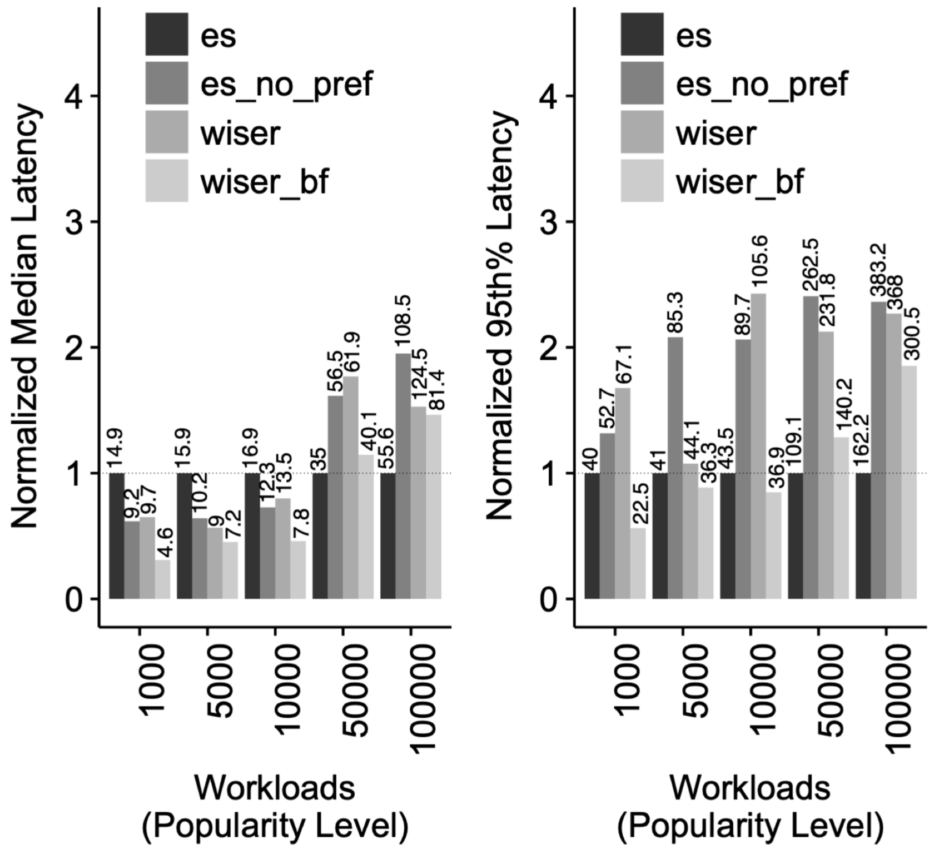
End-to-end Performance

Phrase Query QPS



- Throughput: QPS (Queries Per Second)
- es: Elasticsearch with prefetch (128 KB)
- es_no_pref: Elasticsearch without prefetch
- wiser: WiSER without Bloom filters
- wiser_bf: WiSER with Bloom filters

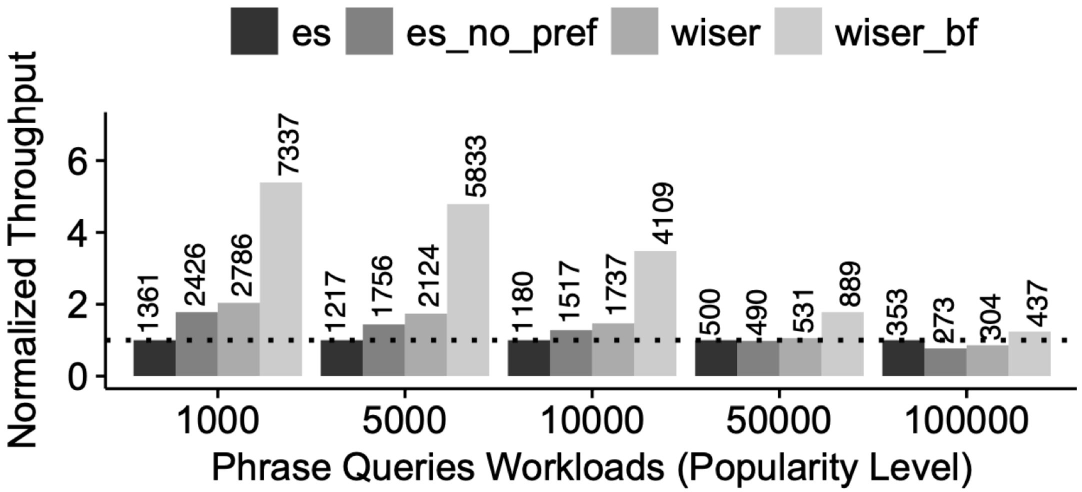
Phrase Queries Latency



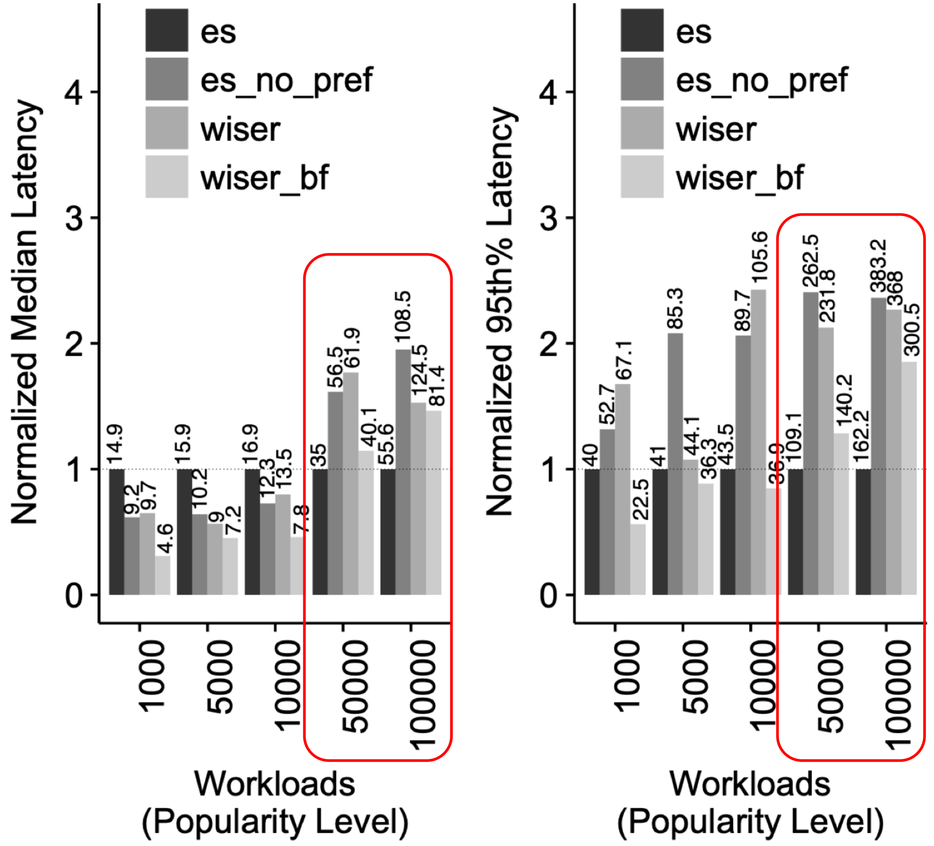
End-to-end Performance

USTC, CHINA ADSLAB

Phrase Query QPS



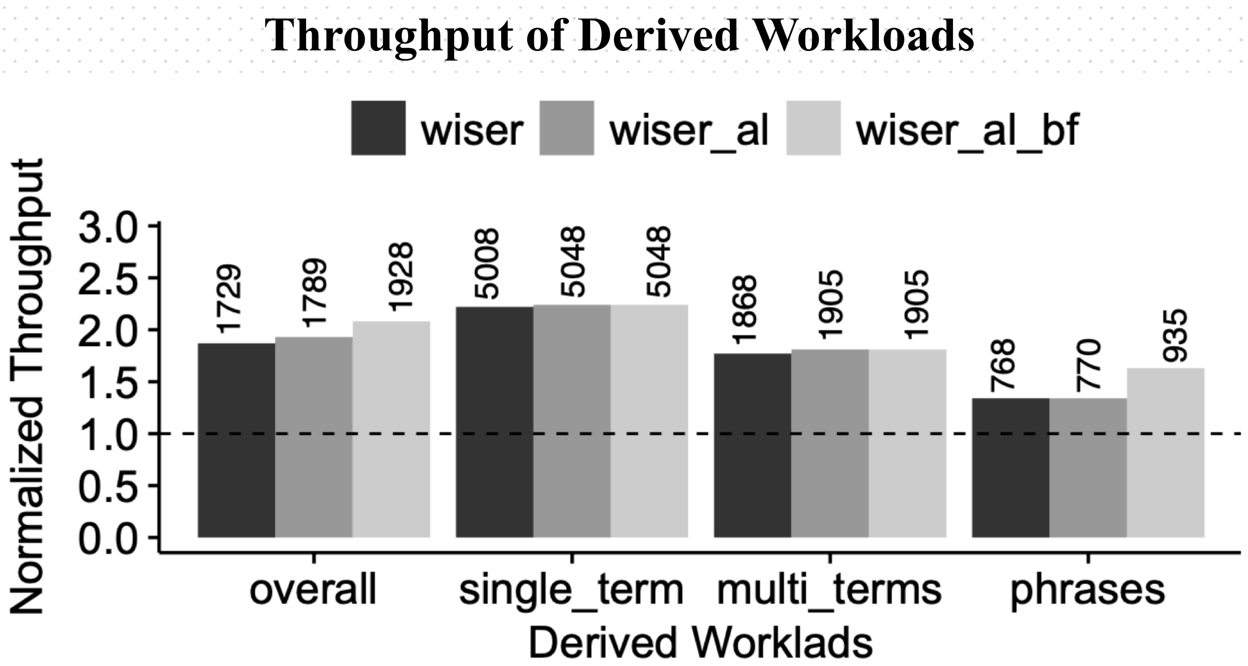
Phrase Queries Latency



- Elasticsearch with OS prefetch (es) achieves the lower latency because the OS prefetches 128 KB of positions data and avoids waiting for many page faults.
- Although the latency of individual queries is lower, the query throughput is also lower due to the read amplification caused by prefetch.

End-to-end Performance

USTC, CHINA ADSLAB



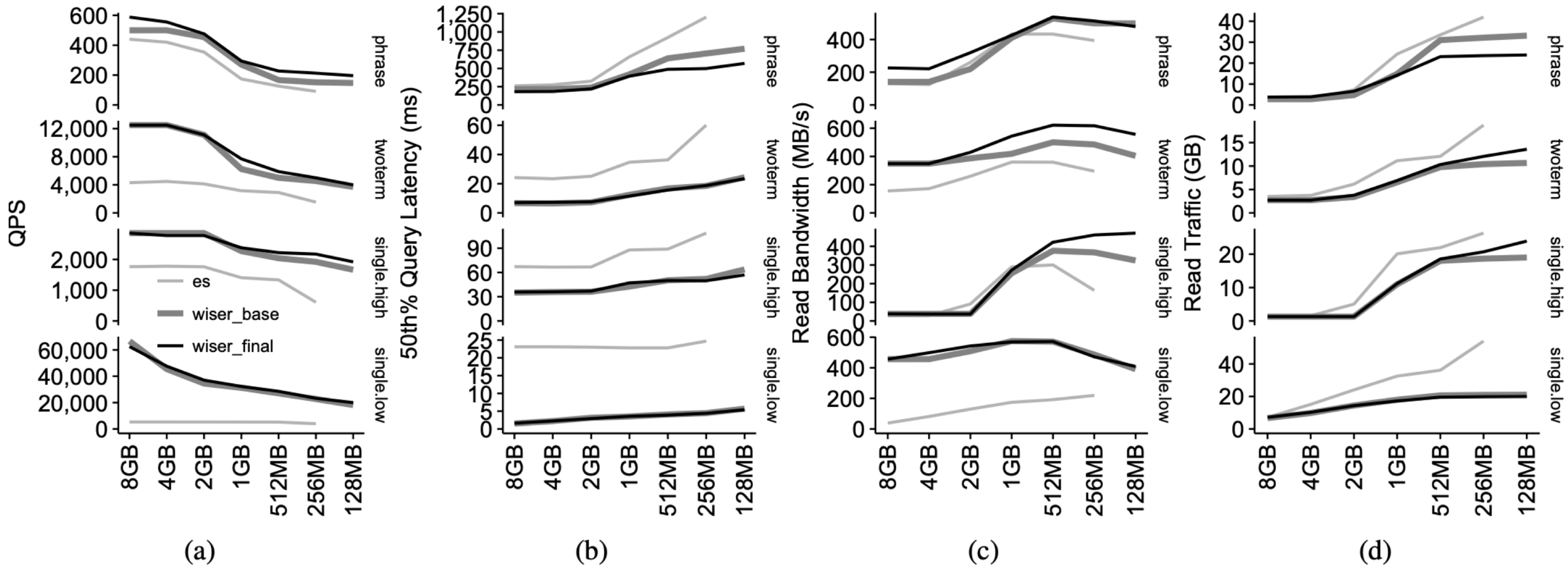
- wiser: unknown
- wiser_al: unknown
- wiser_al_bf: unknown
- Normalized to the throughput of Elasticsearch without Prefetching

- For single-term queries, WiSER achieves as high as 2.2x throughput compared to Elasticsearch.
  - Around 60% queries in the real workload are of popularity less than 10,000.
- For multi-term match queries, grouped data layout also helps to increase throughput by more than 60%.
- For phrase queries, WiSER with Bloom filters increases throughput by more than 60%.

Scaling with Memory

USTC, CHINA ADSLAB

Performance over a range of memory sizes

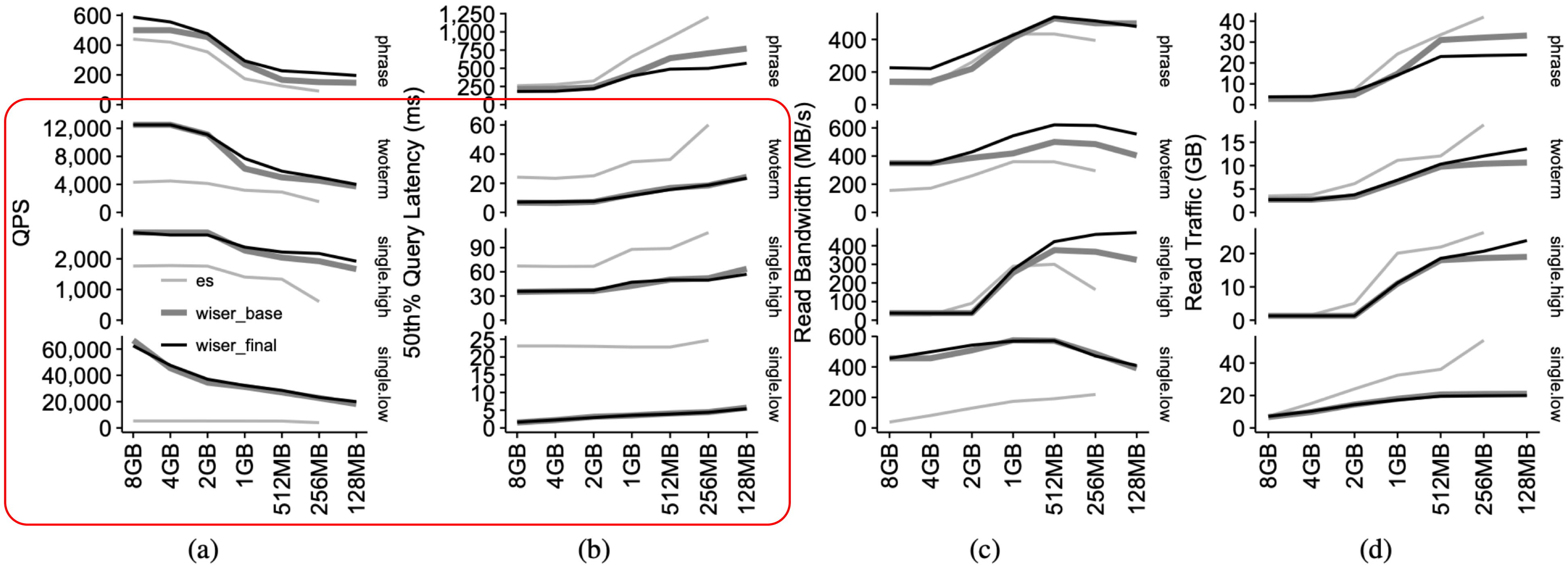


- es: Elasticsearch without prefetch
- wiser_base: WiSER with only cross-stage grouping
- wiser_final: fully-optimized WiSER
- single.high: high popularity level
- single.low: low popularity level

Scaling with Memory

USTC, CHINA ADSLAB

Performance over a range of memory sizes

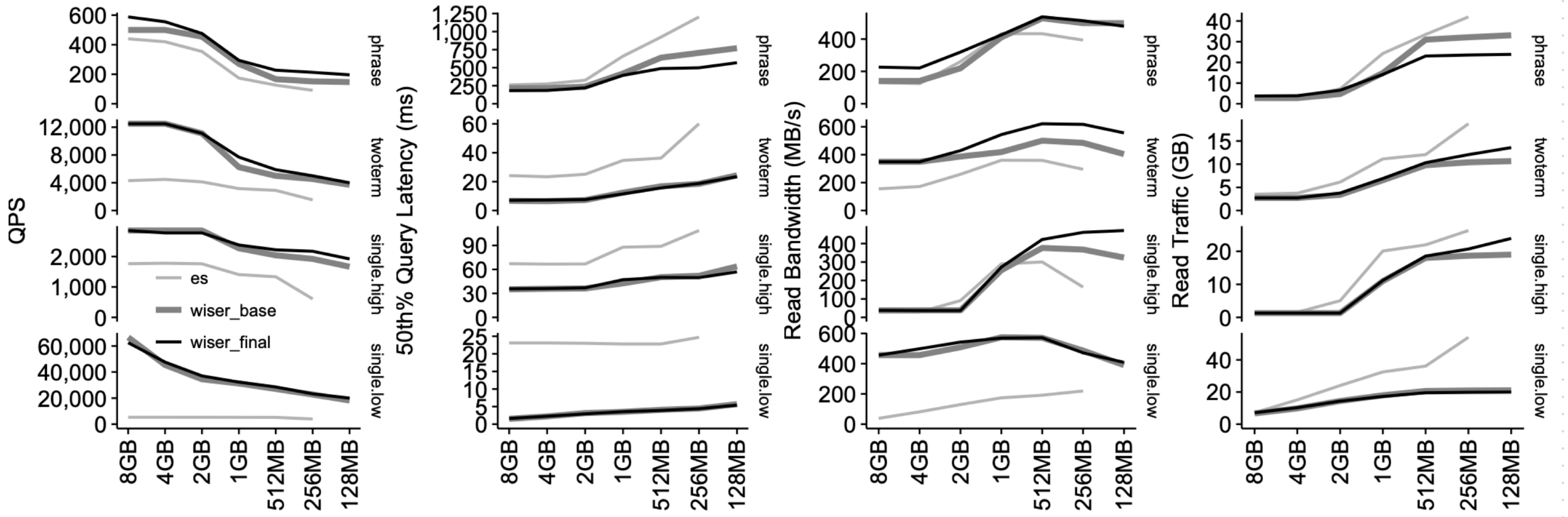


Due to the network issue of Elasticsearch.

Scaling with Memory

USTC, CHINA ADSLAB

Performance over a range of memory sizes



(a) (b) (c) (d)

- As expected, query throughput is higher, and latency is lower, when more memory is available.
- WiSER has much higher query throughput and much lower query latency than Elasticsearch across all workloads and memory sizes.
- WiSER's traffic sizes increase much slower than Elasticsearch's as we reduce memory sizes.

Outline

USTC, CHINA ADSLAB

- Motivation
- Background
- Design
- Implementation
- Evaluation
- **Conclusion**

Conclusion

IO is the bottleneck

Conclusion

IO is the bottleneck

Cross-stage
Data Grouping

Two-Way Cost-Aware
Bloom Filters

Adaptive Prefetching

Trade Disk Space
for Less I/O

Conclusion

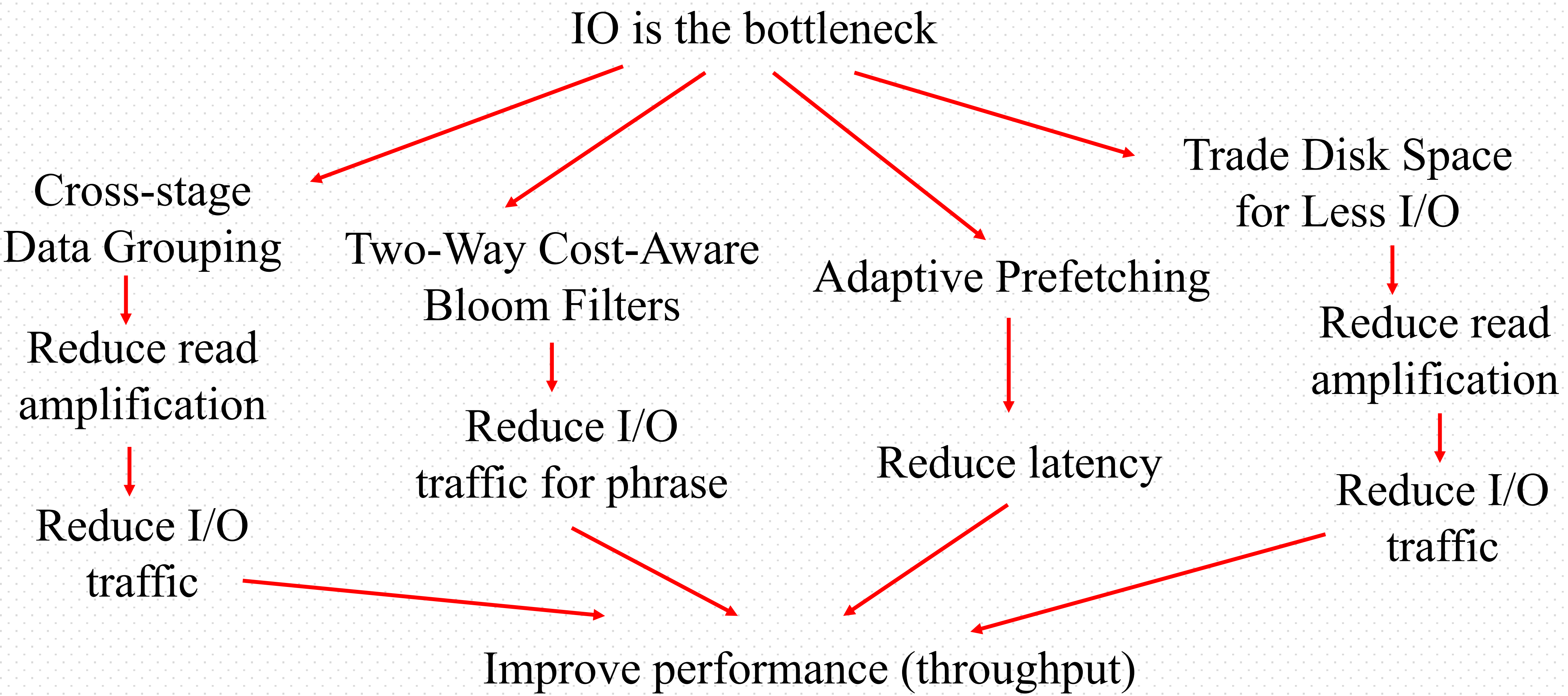
USTC, CHINA ADSLAB

IO is the bottleneck

- Cross-stage Data Grouping → Reduce read amplification → Reduce I/O traffic
- Two-Way Cost-Aware Bloom Filters → Reduce I/O traffic for phrase
- Adaptive Prefetching → Reduce latency
- Trade Disk Space for Less I/O → Reduce read amplification → Reduce I/O traffic

Conclusion

USTC, CHINA ADSLAB



Read as Needed: Building WiSER, a Flash-Optimized Search Engine

Thanks & QA!

USTC, CHINA
ADSLAB
先进数据系统实验室

April 24th, 2020