Think Before You Shuffle

Data-Driven Shuffles for Geo-Distributed Analytics

Maruth Goyal

Background

Background
Joins

UT Students

ID	Name	Age	Major	Year
1	Aditya Parulekar	3	Flexing	Freshman
2	Aditya Arjun	1	Being Bougie	Sophomore
3	Aditya Tewari	-1	Complaining loudly	Graduated thank god
4	Rahul Krishnan	21	Computer Science	Senior

CS439H

ID	Grade
2	C
1	D
4	A+

I want to know the names and years of the students in my class along with their grade!

Know the ID of the students,
Can we just ***join*** it with the IDs in the other table?

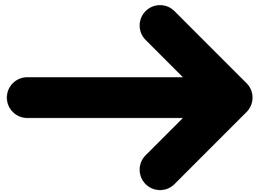
Background
Joins

UT Students

ID	Name	Age	Major	Year
1	Aditya Parulekar	3	Flexing	Freshman
2	Aditya Arjun	1	Being Bougie	Sophomore
3	Aditya Tewari	-1	Complaining loudly	Graduated thank god
4	Rahul Krishnan	21	Computer Science	Senior

CS439H

SID	Grade
2	A-
1	D
4	A+



```
SELECT ID, Name, Year, Grade
FROM
UT Students JOIN CS439H
ON ID = SID
```

ID	Name	Year	Grade
1	Aditya Parulekar	Freshman	D
2	Aditya Arjun	Sophomore	A-
4	Rahul Krishnan	Senior	A+

Background

Joins

JOIN(A, B, $[A_1, \ldots, A_N]$**,** $[B_1, \ldots, B_N]$**)**

outputs all tuples

(r_A, r_B)

where r_A is a row of A and r_B is a row of B,

and $r_A[A_i] = r_B[B_i]$ for all i

```
foreach tuple r ∈ R:
  foreach tuple s ∈ S:
    emit, if r and s match
```

Outer

Inner

R(id,name)

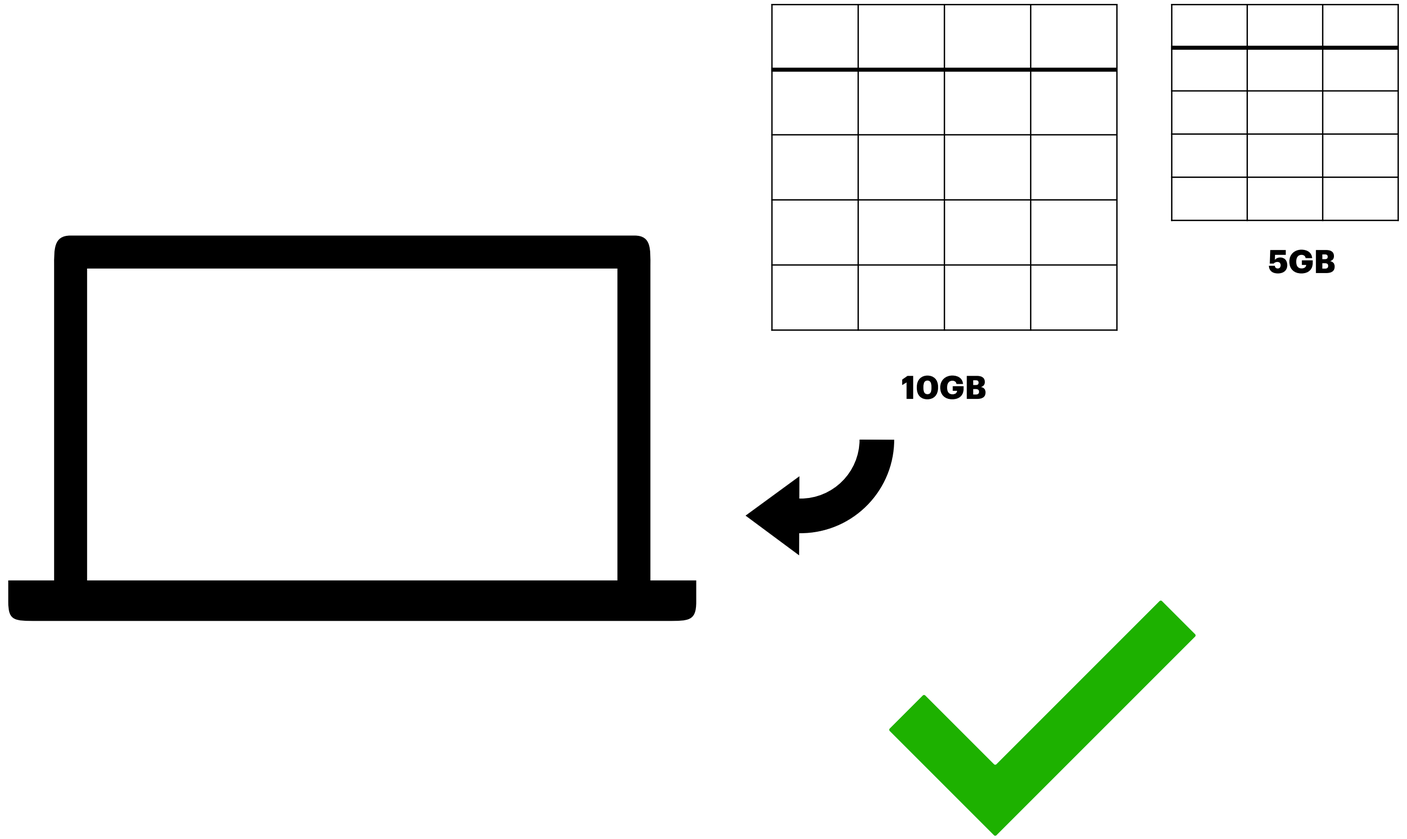
id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/7/2020
500	7777	10/7/2020
400	6666	10/7/2020
100	9999	10/7/2020
200	8888	10/7/2020

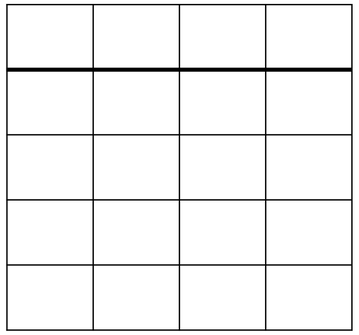
Background

Distributed Query Execution

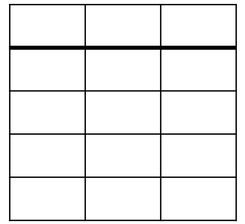


Background

Distributed Query Execution



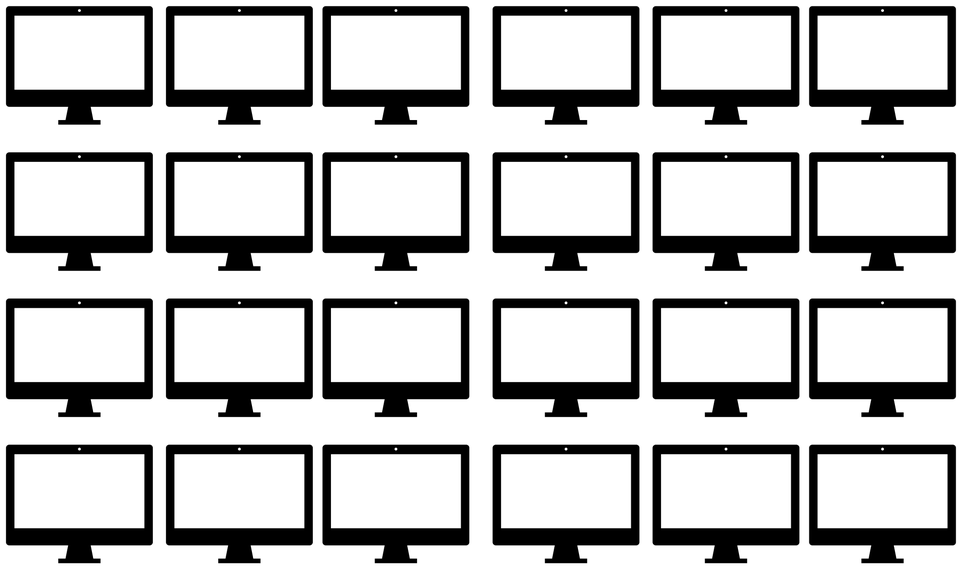
10TB

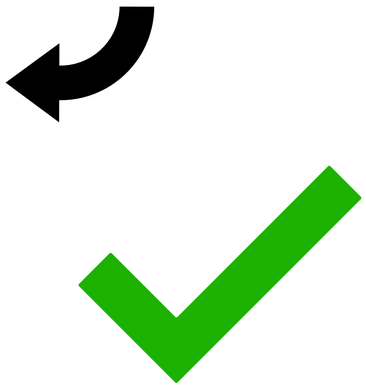


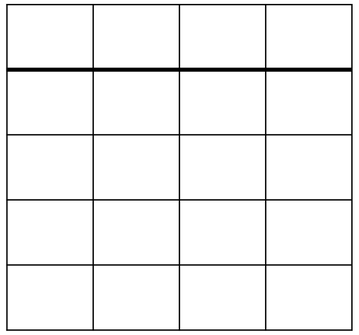
5TB

Background

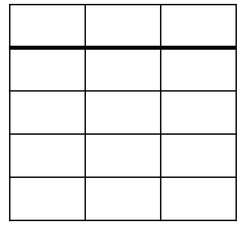
Distributed Query Execution



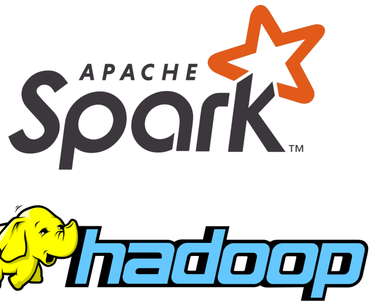




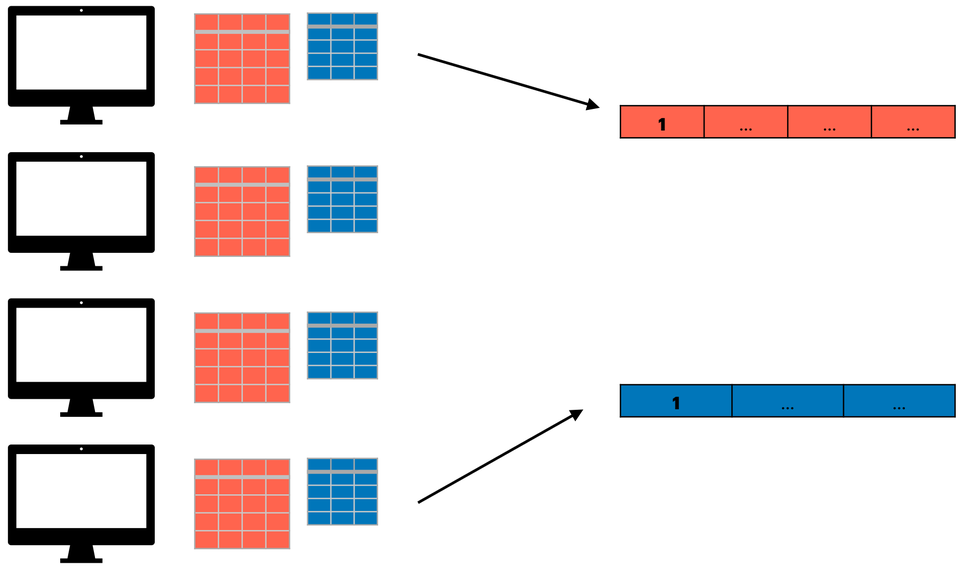
10TB

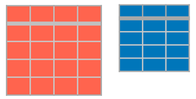


5TB



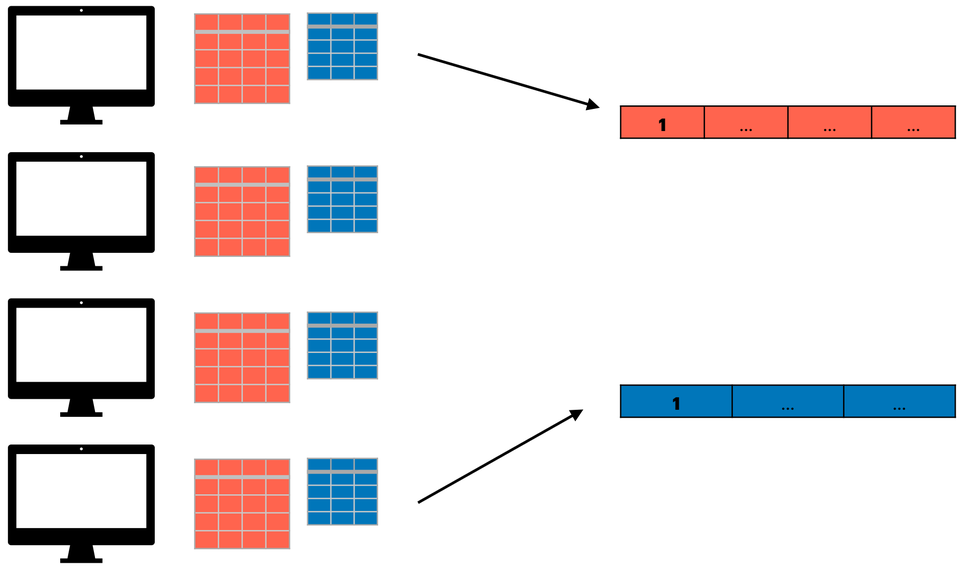
Background
Distributed Query Execution

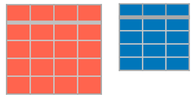


To join the tables, can we just locally join each  and just collect the result?

Might miss rows that should be in join output!

Background
Distributed Query Execution



To join the tables, can we just locally join each  and just collect the result?

Yes! ***If*** we *repartition* the tables so that any rows that may be in join output should be on same machine

Background
Distributed Query Execution

To join the tables, can we just locally join each  and just collect the result?

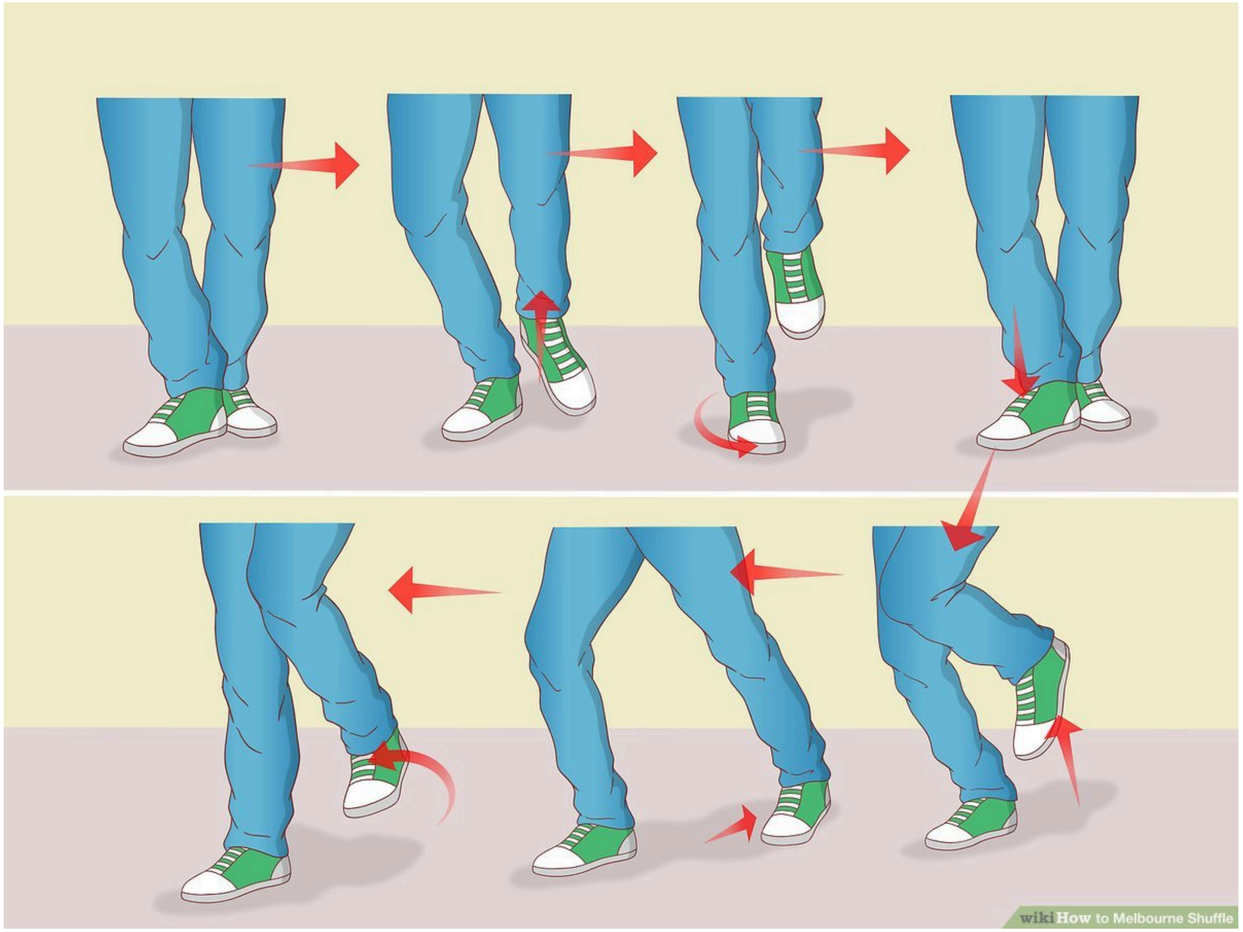
Yes! ***If*** we *repartition* the tables so that any rows that may be in join output will be on same machine

"shuffling"

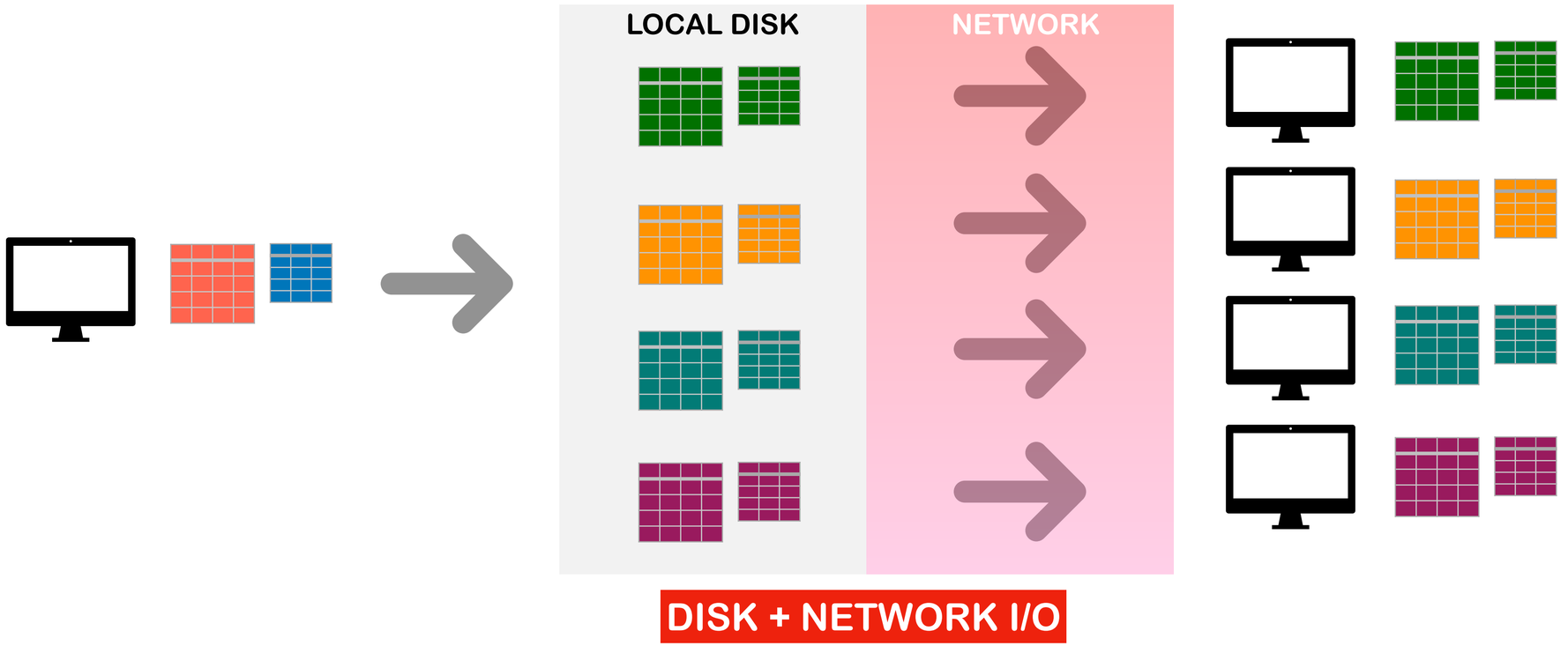
Hash Partitioning: Partition by hash(keys) mod #machines

Background

How to Shuffle?

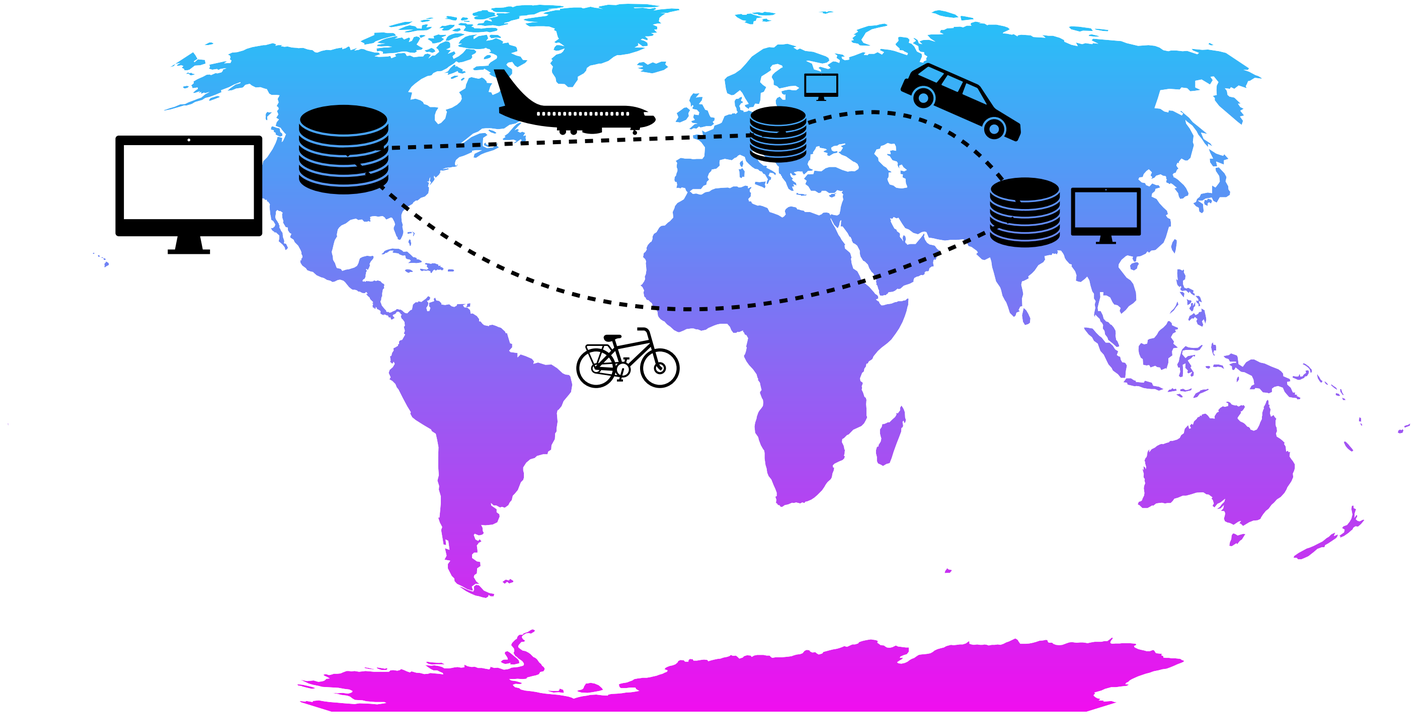


Background
Shuffle

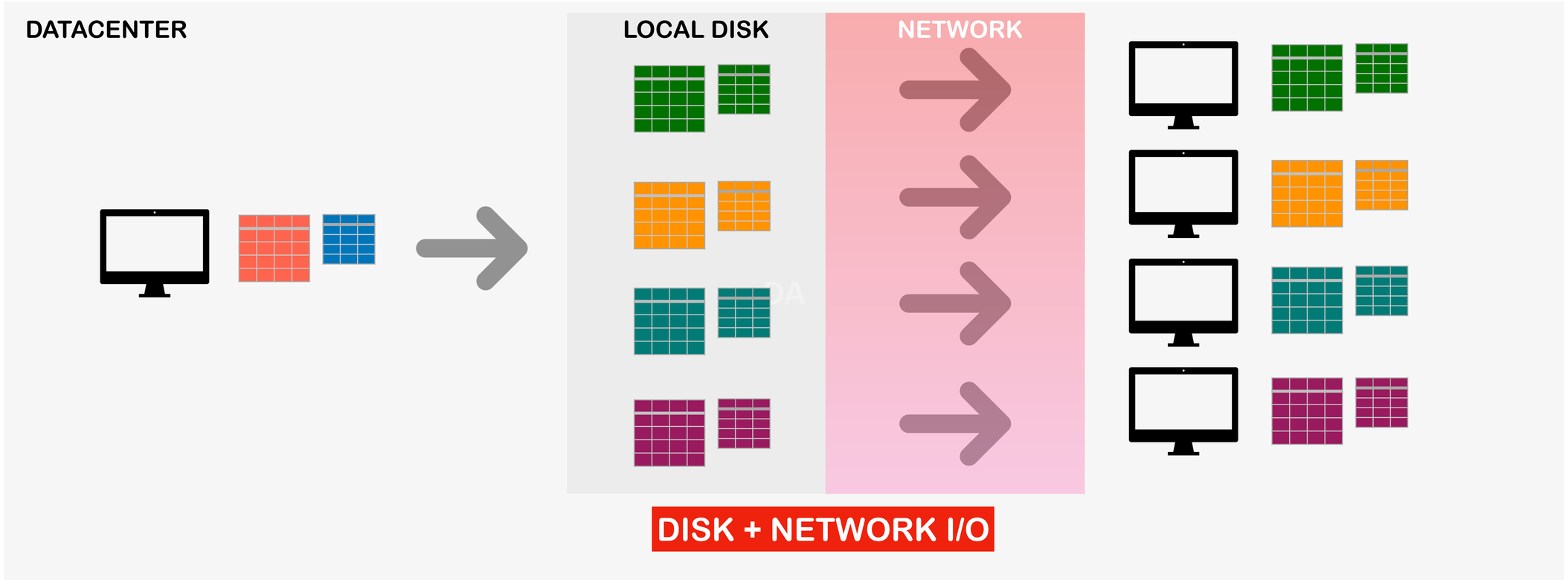


Motivation

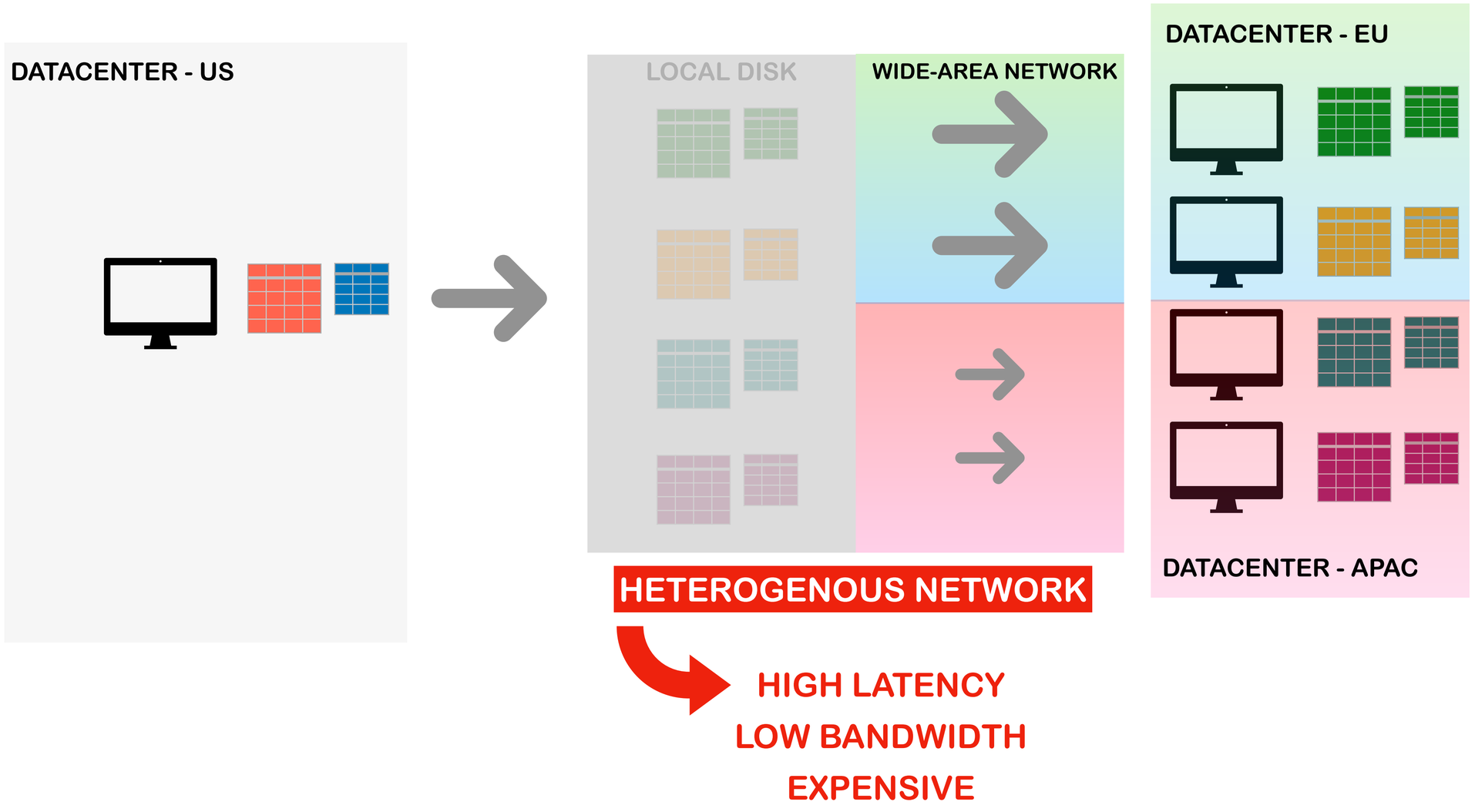
Geo-Distributed Analytics



- Data Locality Legislation
- Store/Compute at edge for performance


DATACENTER
LOCAL DISK
NETWORK
DISK + NETWORK I/O

Geo-Distributed Analytics



Need to minimize amount of data sent over network

Technique

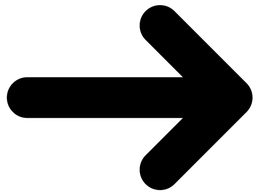
Key Observation

UT Students

ID	Name	Age	Major	Year
1	Aditya Parulekar	3	Flexing	Freshman
2	Aditya Arjun	1	Being Bougie	Sophomore
3	Aditya Tewari	-1	Complaining loudly	Graduated thank god
4	Rahul Krishnan	21	Computer Science	Senior

CS439H

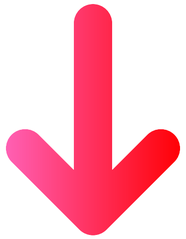
SID	Grade
2	A-
1	D
4	A+



```
SELECT ID, Name, Year, Grade
FROM
UT Students JOIN CS439H
ON ID = SID
```

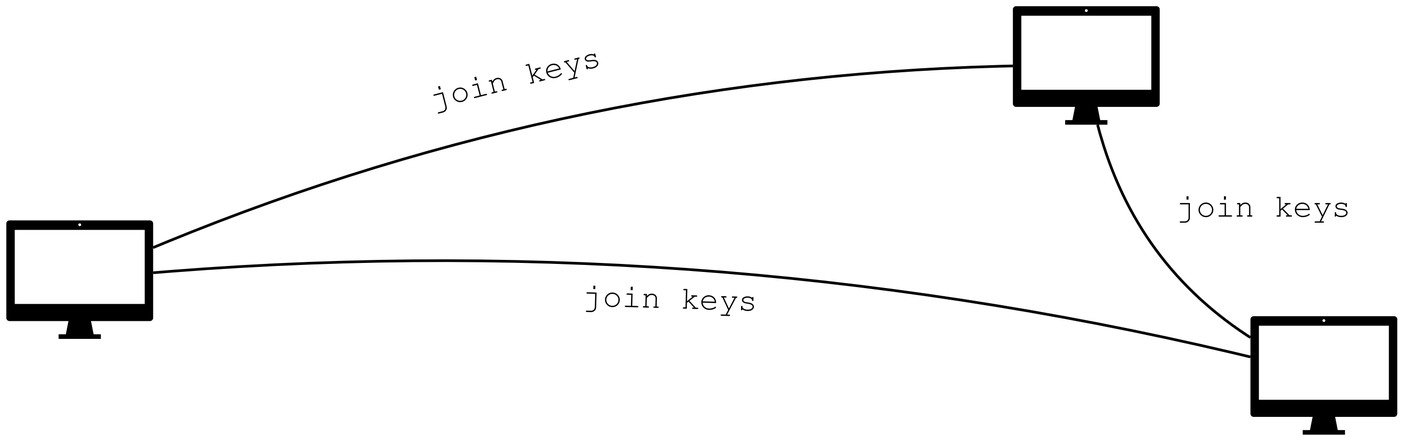
ID	Name	Year	Grade
1	Aditya Parulekar	Freshman	D
2	Aditya Arjun	Sophomore	A-
4	Rahul Krishnan	Senior	A+

Only a subset of rows will be in the join output



We're sending useless rows during shuffles!

Strawman Solution

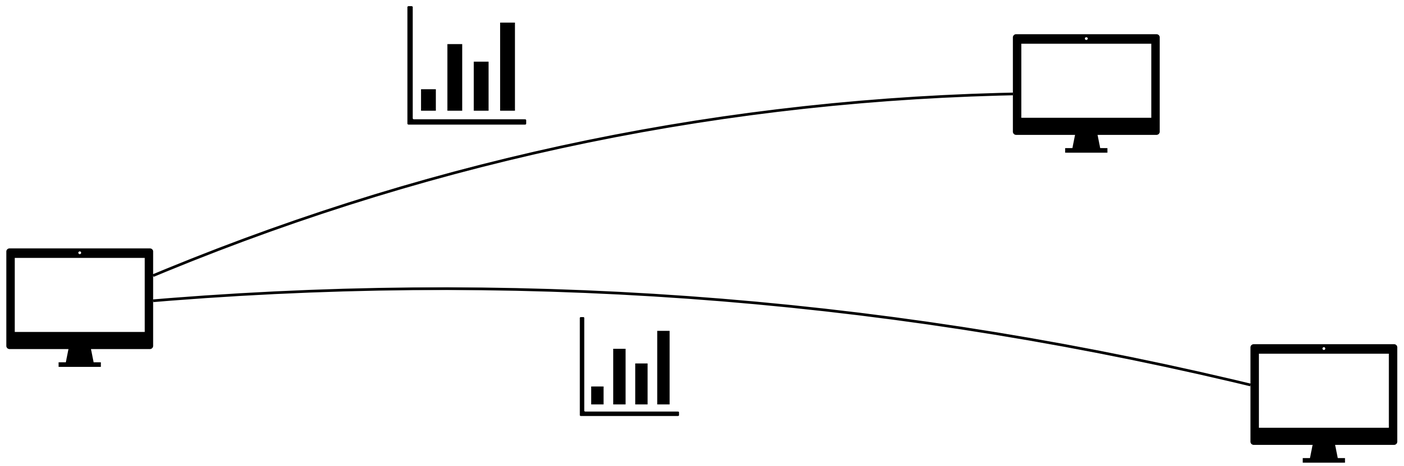


-Broadcast Join Keys
-Compute Intersection
-Only send rows with key in intersection

- size(join keys) ~ size(rows)
- 100 TB table with 5 cols, unique join key ~ 20TB

+Optimal # of rows transferred!

Alternative Strawman Solution



-Send histogram of keys in partition
-Identify disjoint bins, tell other machines
-Only send rows with key not in disjoint bin

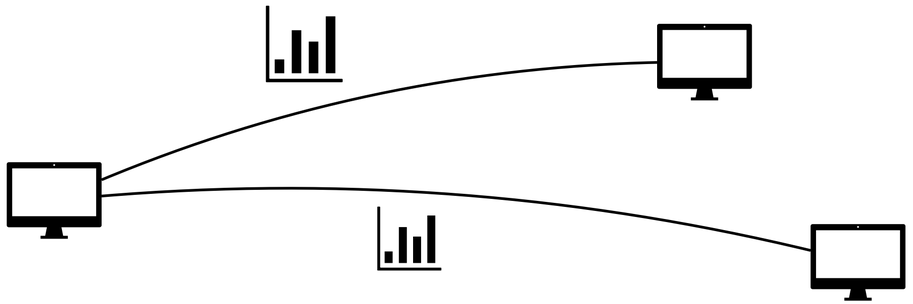
- Histogram with low #bins is very coarse-grained
  -Low likelihood of finding disjoint bins

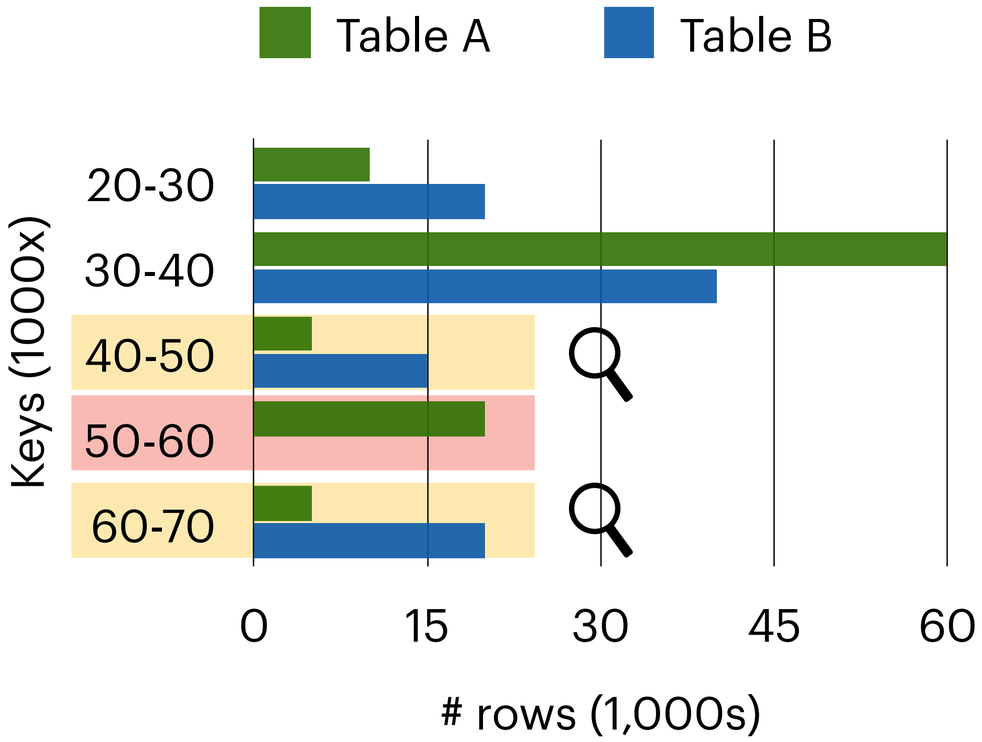
+ Compact!

Design Goals

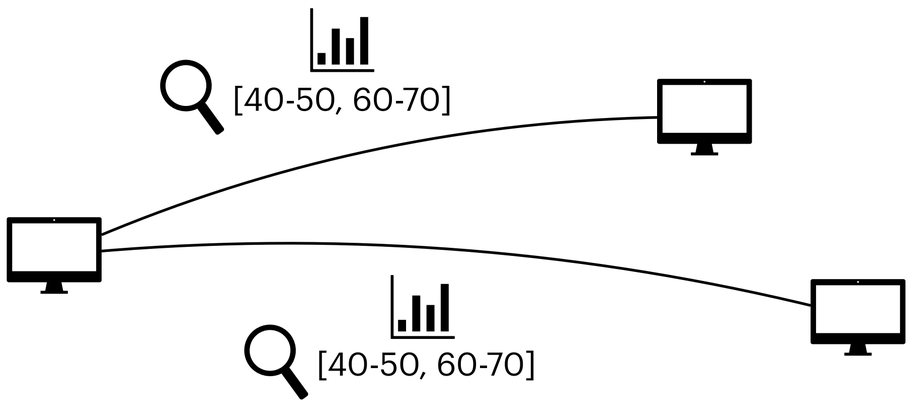
+ Closely approximate optimal rows
+ Utilize fine-grained information
+ Information should be compact

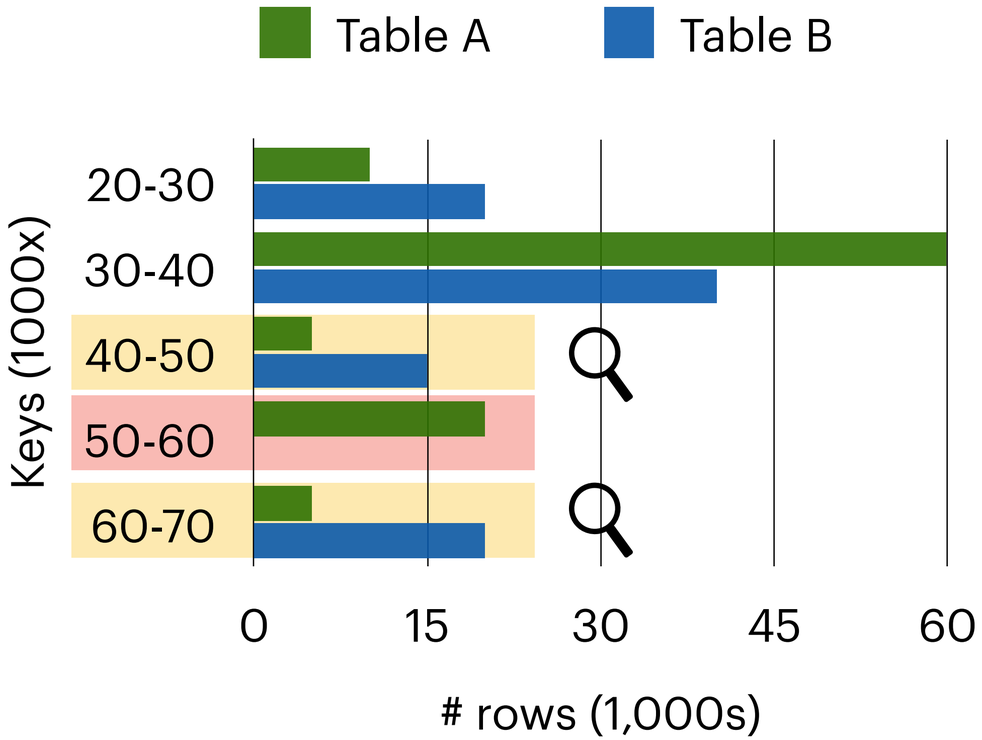
Solution By Example



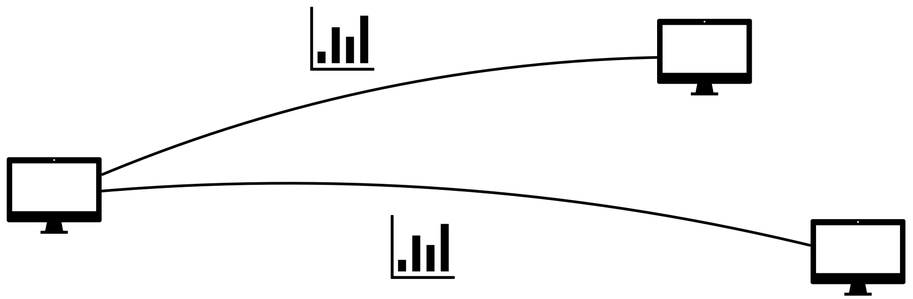


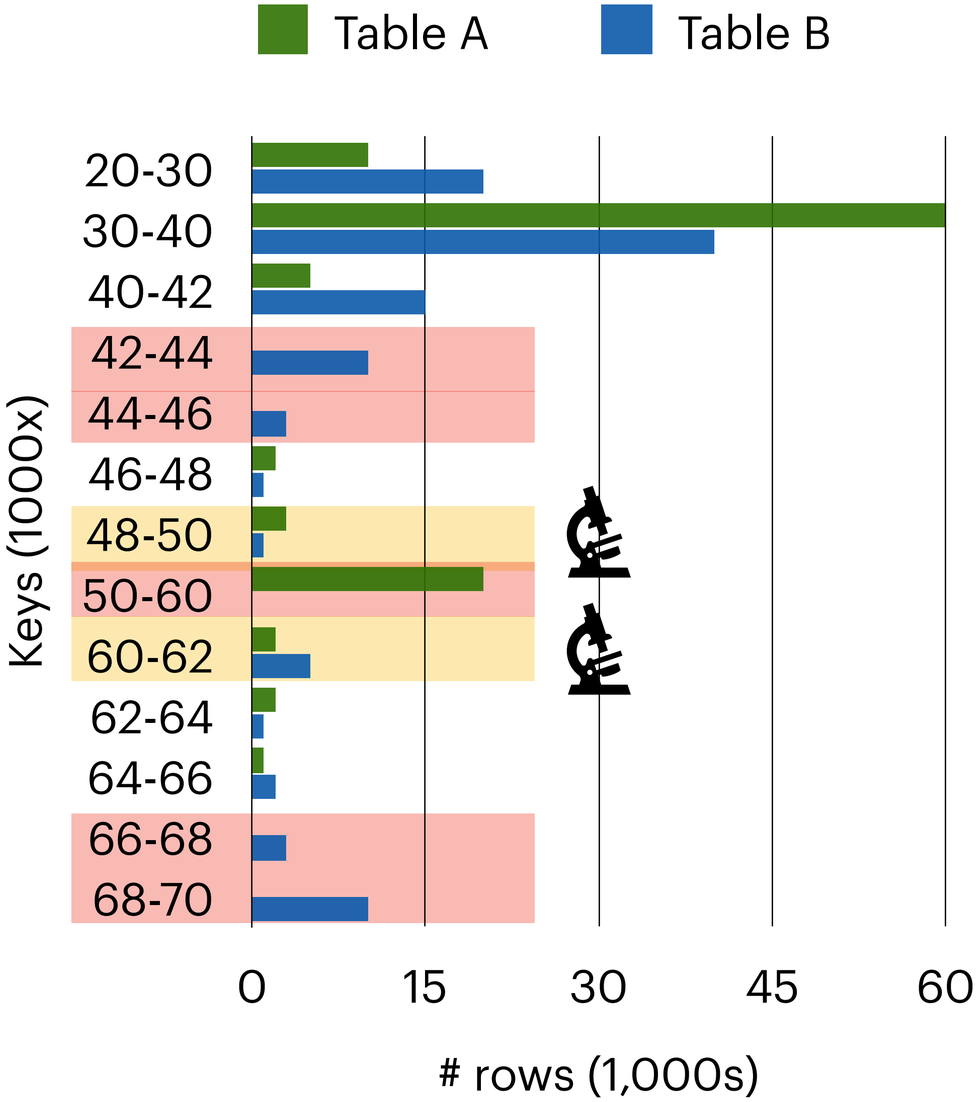
Solution By Example



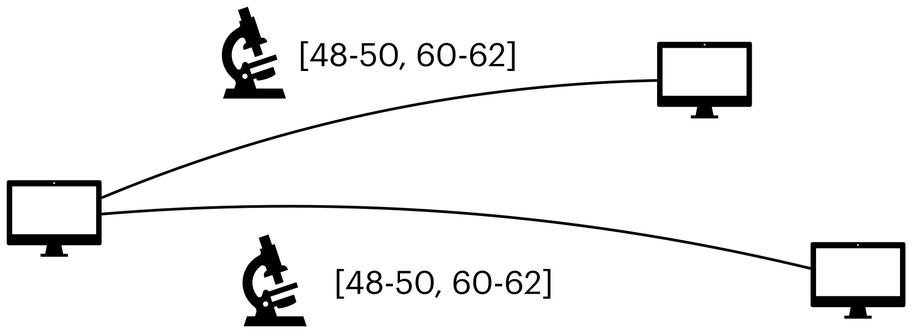


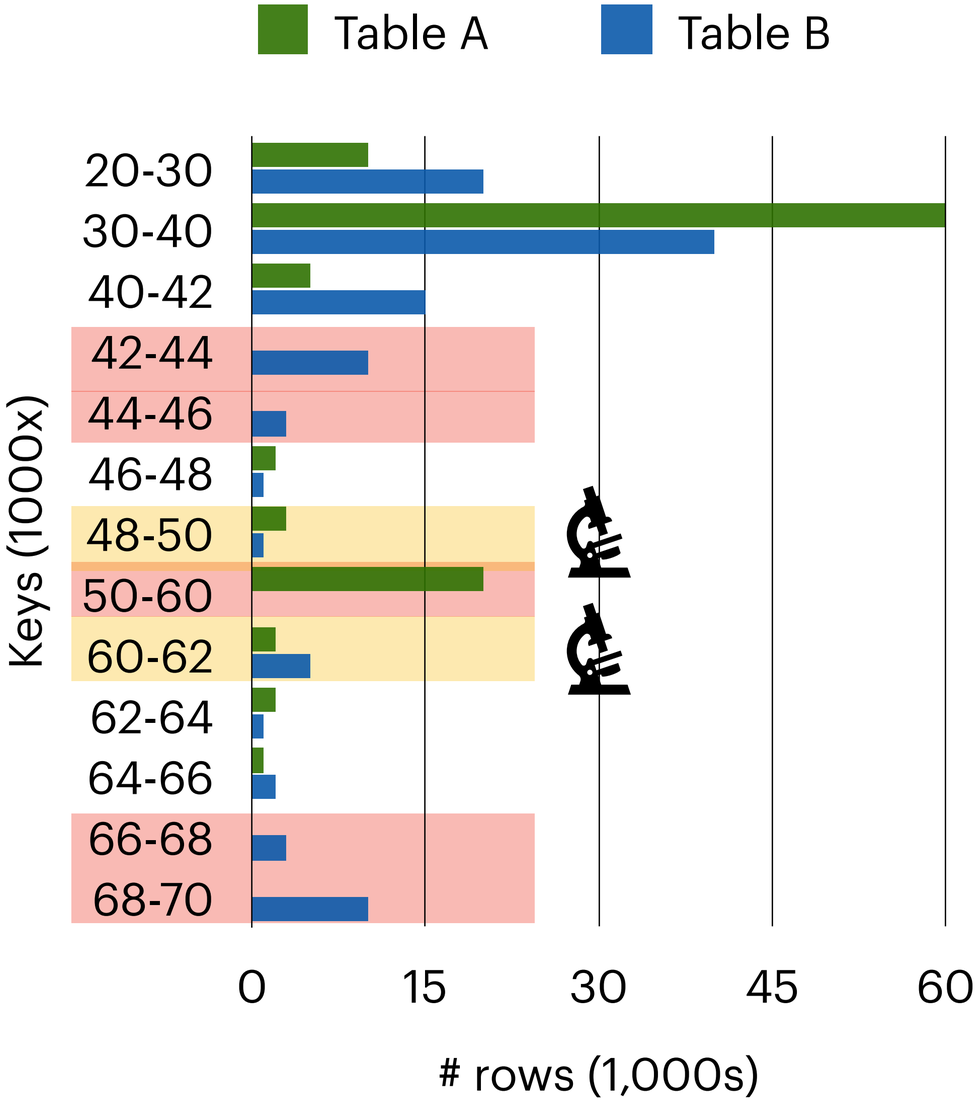
Solution By Example



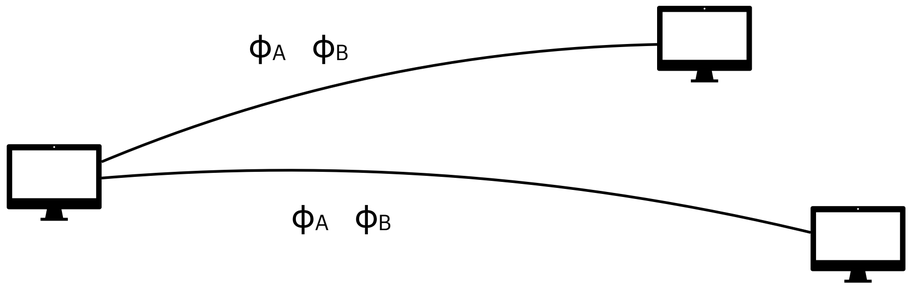


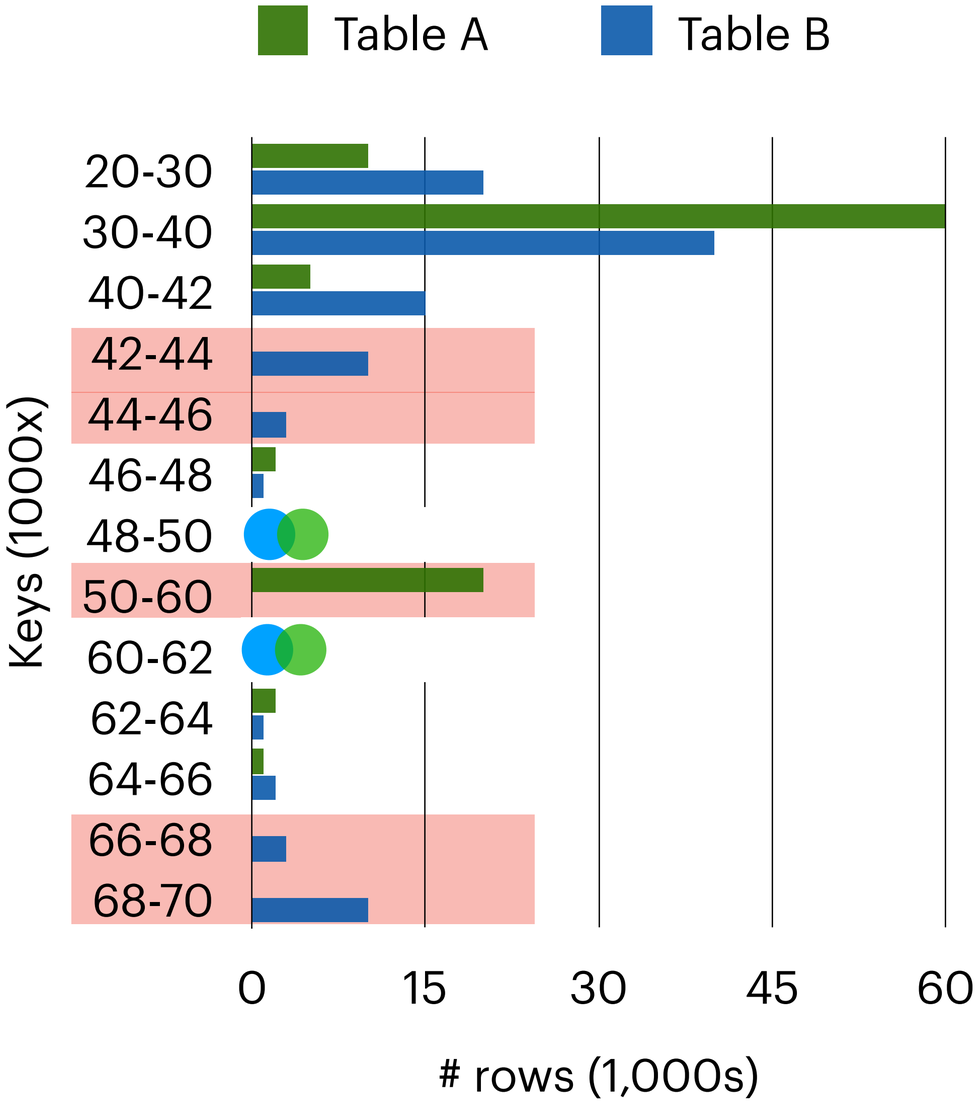
Solution By Example





Solution By Example



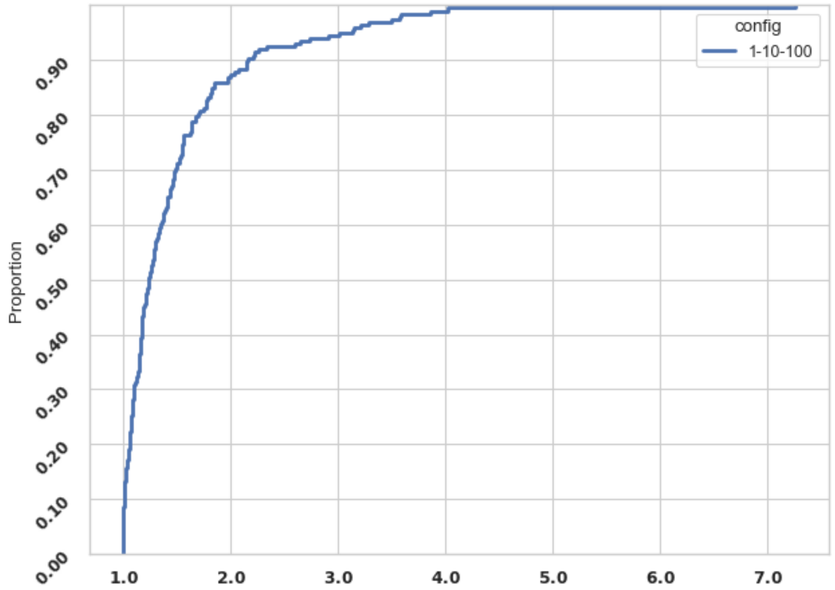


Evaluation

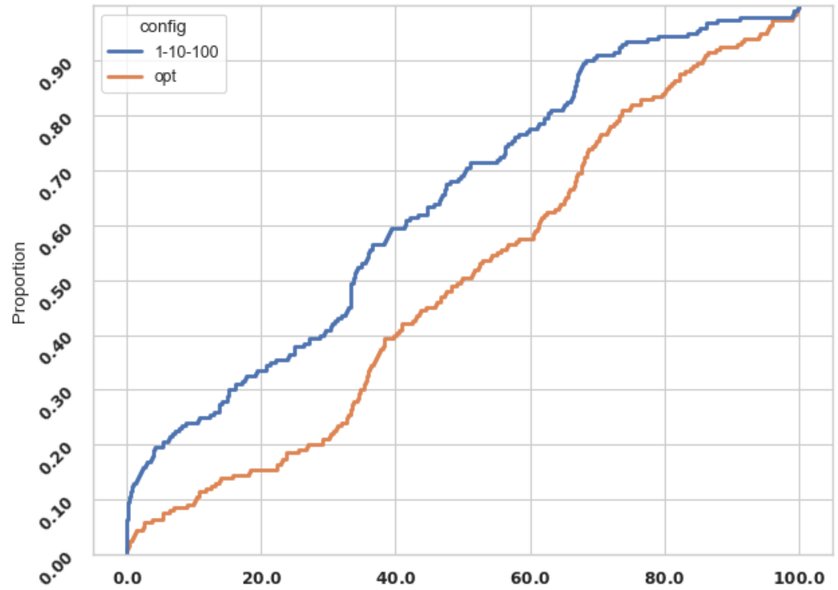
Benchmark Setup

- **Synthetic Data:** Sample number of rows from $N(10^5, 10^4)$ and $N(2 \times 10^4, 500)$ for left and right table respectively.
  - Join key sampled from mixture of 5 Gaussians $N(m, s)$ where $m \sim U[0, 10{,}000]$, $s \sim U[0, 1000]$.
- **TPC-DS**: Standard benchmark for data analytics. Extracted the first inner equi-join from queries which contained at least one, materialized inputs to get left and right table
- Assume 1 consumer, 2 producers

Synthetic Benchmark



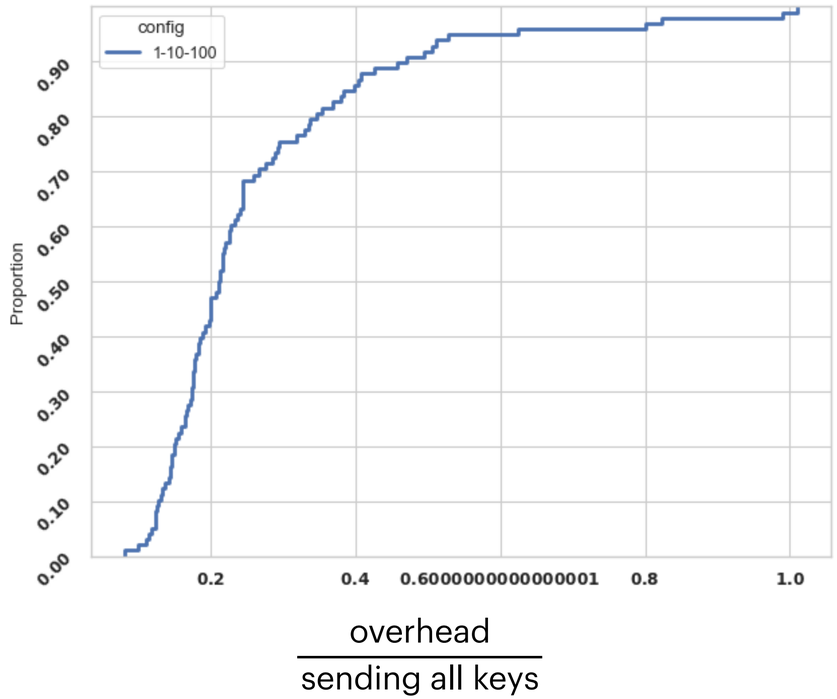
$$\frac{\text{number of rows sent}}{\text{optimal number of rows}}$$



Percentage reduction in #rows sent

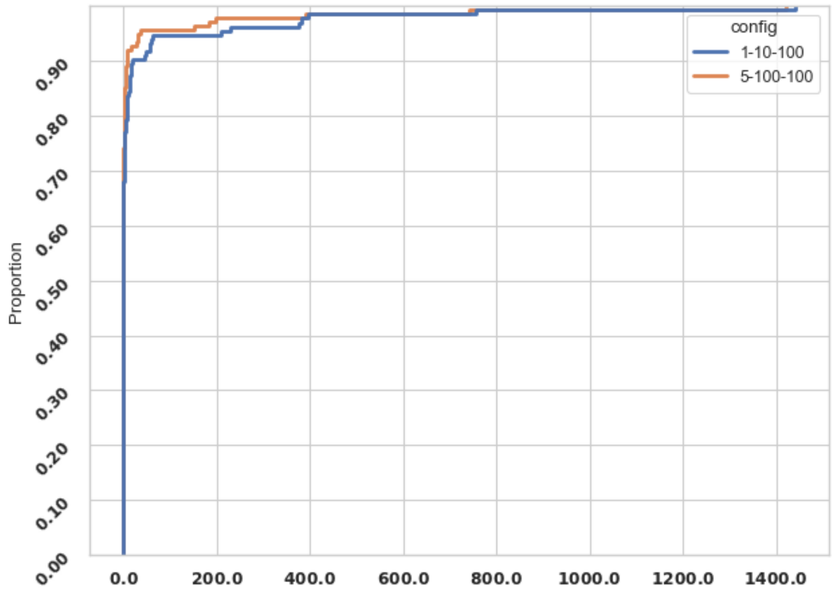
1. For **60%** queries sent **<=1.3x** the optimal number of rows. For **80% <=1.7x**
2. For **60%** queries **>30%** reduction in number of rows sent. For **30% >50%** reduction.
3. Config: 1 bin examined per round, histograms have 10 bins, request distinct keys if bin width is at most 100. Number of rounds is $2 \cdot \log_{10}(\text{range})$

Synthetic Benchmark

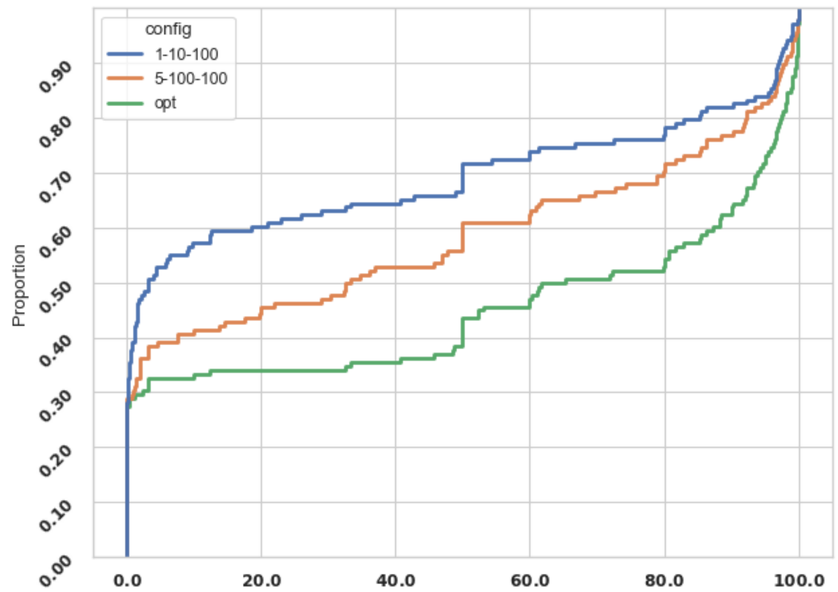


1. For **70%** of queries, **>=4x** smaller overhead
2. For **50%** of queries **>=5x** smaller

TPC-DS



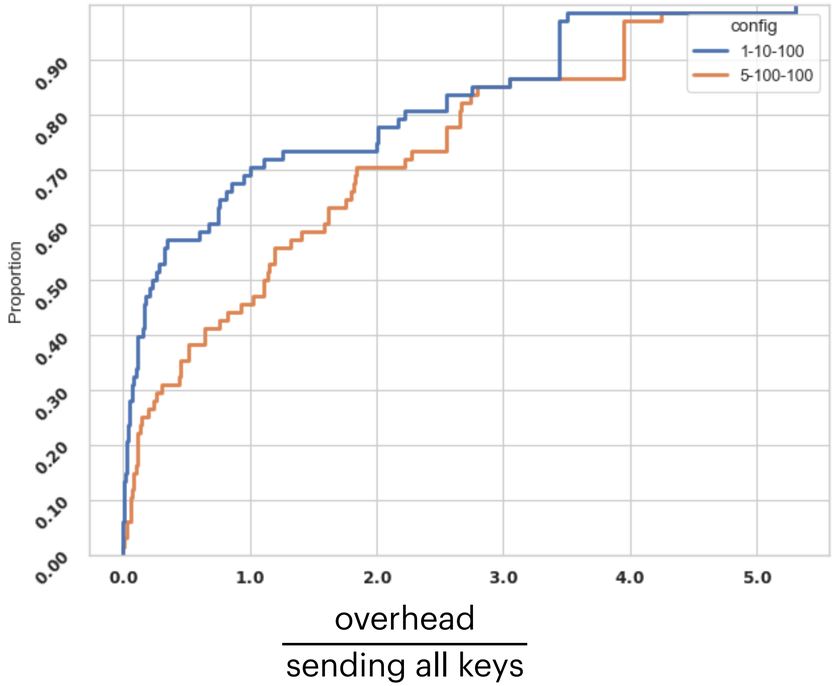
$$\frac{\text{number of rows sent}}{\text{optimal number of rows}}$$



Percentage reduction in #rows sent

1. For **80%** queries sent **<=1.9x** the optimal number of rows. For **70% match**
2. For **55%** queries **>30%** reduction in number of rows sent. For **40% >50%** reduction.
3. Config: 5 bins examined per round, histograms have 100 bins, request distinct keys if bin width is at most 100. Number of rounds is $2 \cdot \log_{10}(\text{range})$
4. 1-10-100 config sent **<=1.14x** optimal number of rows for **67.4%** queries, **<=7x** for 80%

TPC-DS



1. For **45%** of queries, smaller overhead
2. For **36%** of queries **>=2x** smaller
3. For **30%** of queries **>=3x** smaller

1. 1-10-100 config, for **70%** queries smaller overhead
2. For **57%** of queries **>=3x** smaller
3. For **50%** queries **>=5x** smaller

Conclusion

Conclusion

- Reducing network-load of shuffles is important in the geo-distributed setting.
- We present a technique that is able to significantly reduce the number of rows sent over the network using run-time fine-grained properties of data with a low overhead in a non-trivial number of cases.
- The technique does not always provide a benefit
  - Much like join algorithms, a query optimizer may decide when it is a good idea to use it

Future Work

Future Work

- **Data-Driven Shuffles for Approximate Analytics**
  - Here, dropping rows is (as such) permissible, can utilize approximate statistics, unsound inference etc.
- **Network-aware aggregation**
  - Instead of everyone sending their statistics to the consumer for merging, send to peers and merge along the way (routing to minimize delay).

Acknowledgements

Acknowledgements