Querying Large Databases

Nathan Beneke

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

15 April 2018
UMM, Morris

The big picture

- Large databases are difficult to query in a reasonable amount of time
- Querying often searches through an excessive amount of data that is "nonsense" or irrelevant
- You don’t always need the exact answer

Outline

Outline

Relational databases display data as *tables*, or *relations* and make it easy to see relationships between different tables

Pet Owners

ID	name
1	Jane
2	John

Pets

ID	ownerID	species	name
1	1	dog	Max
2	1	dog	Tucker
3	1	cat	Athena
4	2	dog	Goldie

Each table is a set of *rows* or *tuples* consisting of the same *attributes*

Queries are questions about databases

- They can be expressed in English
- Formally they can be expressed in SQL or relational algebra

"What are the names of all dogs and their corresponding owners?"

dogName	ownerName
Max	Jane
Tucker	Jane
Goldie	John

Cartesian Product

```
SELECT pets.name, petOwners.name FROM petOwners JOIN
pets WHERE pets.ownerID=petOwners.ID AND
species="dog";
```

"What are the names of all dogs and their corresponding owners?"

petOwners.ID	petOwners.name	pets.ownID	pets.name	species
1	Jane	1	Max	dog
1	Jane	1	Tucker	dog
1	Jane	1	Athena	cat
1	Jane	2	Goldie	dog
2	John	1	Max	dog
2	John	1	Tucker	dog
2	John	1	Athena	cat
2	John	2	Goldie	dog

$n * m$ tuples

Cartesian Product

```
SELECT pets.name, petOwners.name FROM petOwners JOIN
pets WHERE pets.ownerID=petOwners.ID AND
species="dog";
```

"What are the names of all dogs and their corresponding owners?"

petOwners.ID	petOwners.name	pets.ownID	pets.name	species
1	Jane	1	Max	dog
1	Jane	1	Tucker	dog
1	Jane	1	Athena	cat
1	Jane	2	Goldie	dog
2	John	1	Max	dog
2	John	1	Tucker	dog
2	John	1	Athena	cat
2	John	2	Goldie	dog

$n * m$ tuples

Cartesian Product

```
SELECT pets.name, petOwners.name FROM petOwners JOIN
pets WHERE pets.ownerID=petOwners.ID AND
species="dog";
```

"What are the names of all dogs and their corresponding owners?"

petOwners.ID	petOwners.name	pets.ownID	pets.name	species
1	Jane	1	Max	dog
1	Jane	1	Tucker	dog
1	Jane	1	Athena	cat
1	Jane	2	Goldie	dog
2	John	1	Max	dog
2	John	1	Tucker	dog
2	John	1	Athena	cat
2	John	2	Goldie	dog

$n * m$ tuples

Cartesian Product

```
SELECT pets.name, petOwners.name FROM petOwners JOIN
pets WHERE pets.ownerID=petOwners.ID AND
species="dog";
```

"What are the names of all dogs and their corresponding owners?"

petOwners.ID	petOwners.name	pets.ownID	pets.name	species
1	Jane	1	Max	dog
1	Jane	1	Tucker	dog
1	Jane	1	Athena	cat
1	Jane	2	Goldie	dog
2	John	1	Max	dog
2	John	1	Tucker	dog
2	John	1	Athena	cat
2	John	2	Goldie	dog

$n * m$ tuples

Outline

For queries we use a notation similar to mathematical functions

- Q denotes a specific query
- D represents a set of data.
- $Q(D)$ denotes the results of Q on a dataset D
- Both Q and $Q(D)$ have nothing to do with the process used to execute the query

We also use a notation for relationships between attributes

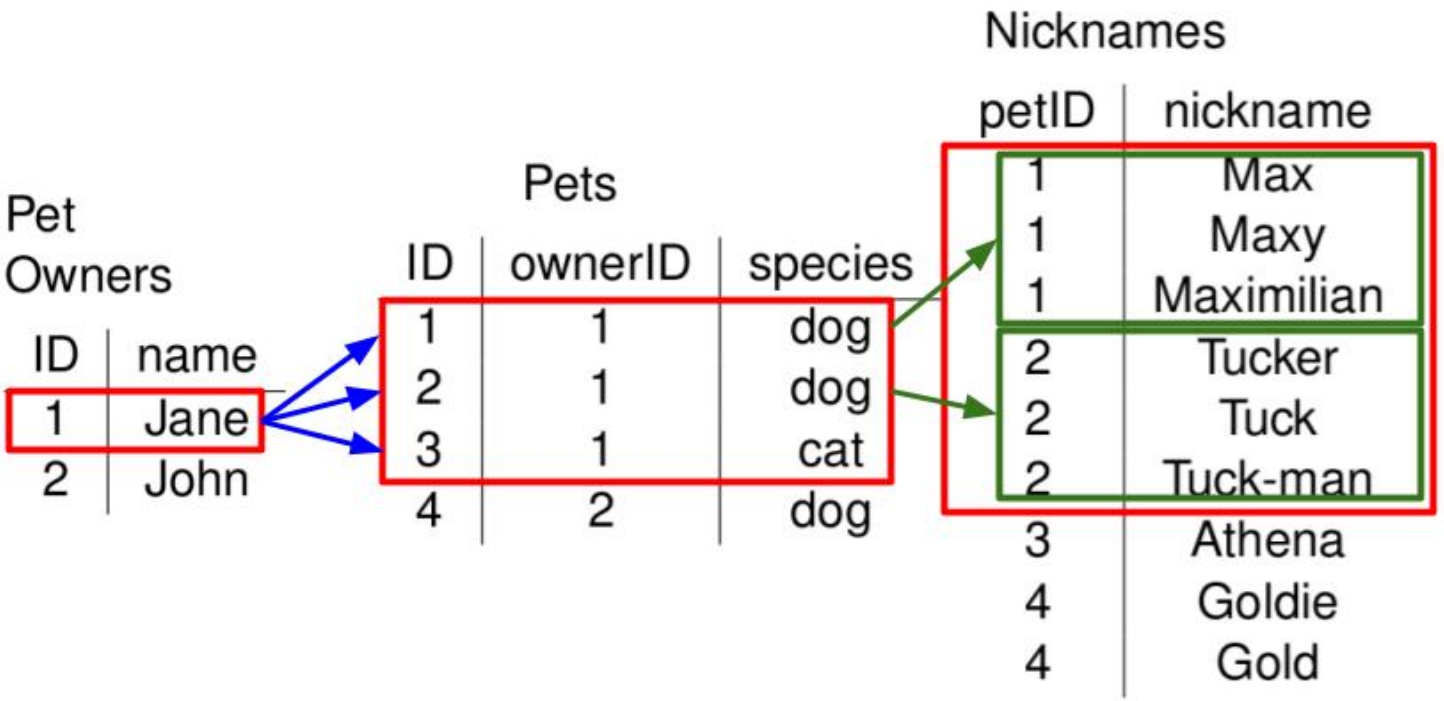
- $R(\{petOwners.ID\} \rightarrow \{pets.ID, pets.name, pets.species\}, 20)$
- $R(X \rightarrow Y, N)$ is a relationship
- X and Y are sets of attributes
- N is a positive integer
- One tuple with X attributes is related to at most N tuples with attributes Y

Scale Independence and Bounded Queries

Q is *scale independent*, w.r.t. a bound $M \geq 0$, in D if there exists a $D_0 \subseteq D$ where $|D_0| \leq M$ such that $Q(D_0) = Q(D)$

Q is *boundedly evaluable* with a set access constraints, $\mathcal{A}$, of relationships $R(X \rightarrow Y, N)$ such that $Q(D)$ can be evaluated using $\mathcal{A}$, and all N are constant.

"Find the nicknames of all of Jane's dogs"



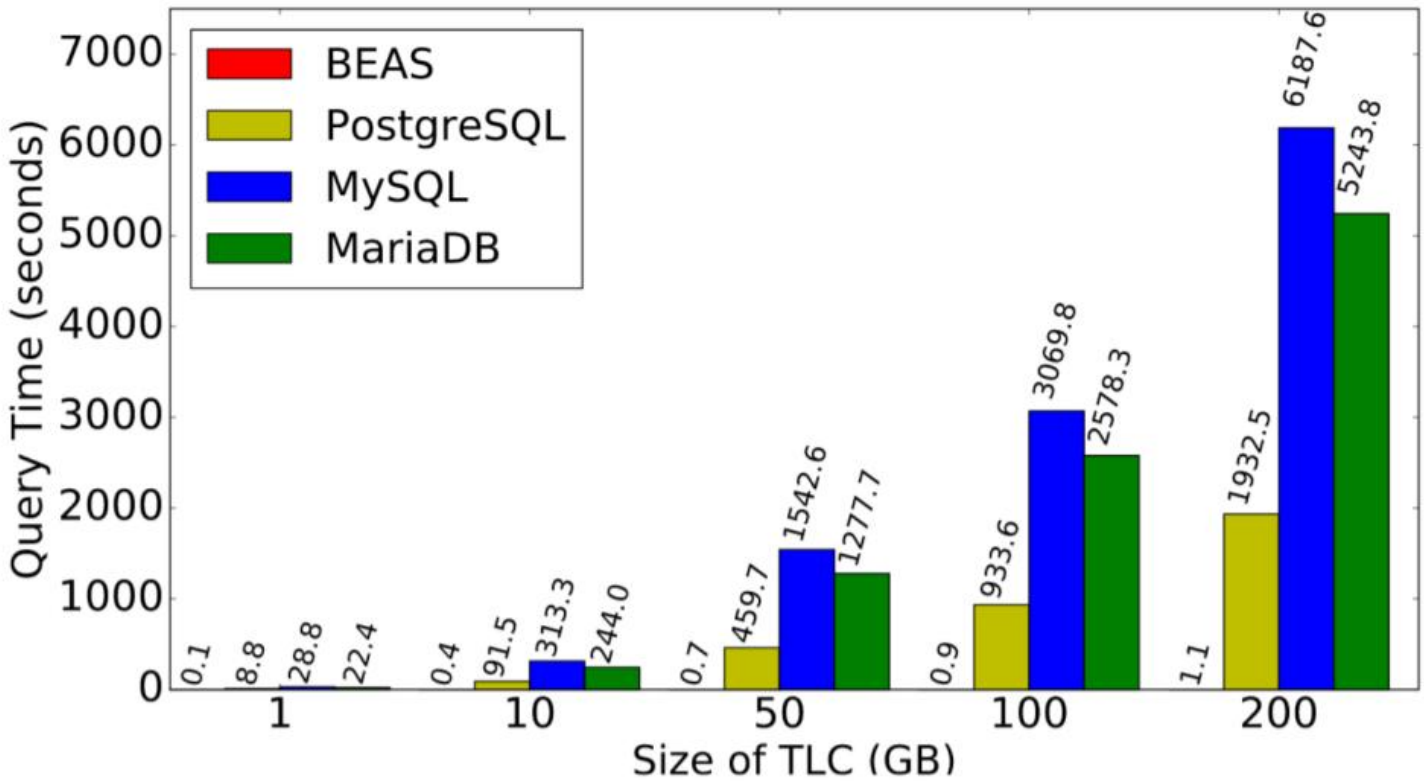
Not all queries are boundedly evaluable, but a larger number of queries have *boundedly evaluable envelopes*

Find boundedly evaluable queries Q_l and Q_u such that $Q_l(D) \subseteq Q(D) \subseteq Q_u(D)$

We can also find *approximation bounds* N_l and N_u such that $|Q(D) - Q_l(D)| \leq N_l$ and $|Q_u(D) - Q(D)| \leq N_u$

Note that Q_l and Q_u depend on D. Q is therefore not boundedly evaluable.

BEAS



Source - BEAS: Bounded Evaluation of SQL Queries

Outline

An aggregate query is a query which asks something about an entire column of another query.

Queries involving functions like average, maximum, or sum

Example: "Find the average age of all dog owners"

Sampling-based Approximate Querying

To evaluate an aggregate query approximately, SAQ runs the query on a random subset of the data.

This works really well for functions like `average`, which are not very sensitive to outliers

SAQ is very inaccurate on functions like `maximum` which are very sensitive to outliers

SAQ works better on data with few outliers like uniform distributions or normal distributions with low variance

Sampling-based Approximate Querying

To evaluate an aggregate query approximately, SAQ runs the query on a random subset of the data.

This works really well for functions like `average`, which are not very sensitive to outliers

SAQ is very inaccurate on functions like `maximum` which are very sensitive to outliers

SAQ works better on data with few outliers like uniform distributions or normal distributions with low variance

Sampling-based Approximate Querying

To evaluate an aggregate query approximately, SAQ runs the query on a random subset of the data.

This works really well for functions like `average`, which are not very sensitive to outliers

SAQ is very inaccurate on functions like `maximum` which are very sensitive to outliers

SAQ works better on data with few outliers like uniform distributions or normal distributions with low variance

Deterministic Approximate Querying

A DAQ execution of a query has the following properties:

- Deterministic error bounds
- Runs for a number of iterations that can be specified
- The accuracy of the results is increased with each iteration
- There is a bounded number of iterations required to achieve an exact result

Deterministic Approximate Querying

A DAQ execution of a query has the following properties:

- Deterministic error bounds
- Runs for a number of iterations that can be specified
- The accuracy of the results is increased with each iteration
- There is a bounded number of iterations required to achieve an exact result

Deterministic Approximate Querying

A DAQ execution of a query has the following properties:

- Deterministic error bounds
- Runs for a number of iterations that can be specified
- The accuracy of the results is increased with each iteration
- There is a bounded number of iterations required to achieve an exact result

Deterministic Approximate Querying

A DAQ execution of a query has the following properties:

- Deterministic error bounds
- Runs for a number of iterations that can be specified
- The accuracy of the results is increased with each iteration
- There is a bounded number of iterations required to achieve an exact result

Deterministic Approximate Querying

A DAQ execution of a query has the following properties:

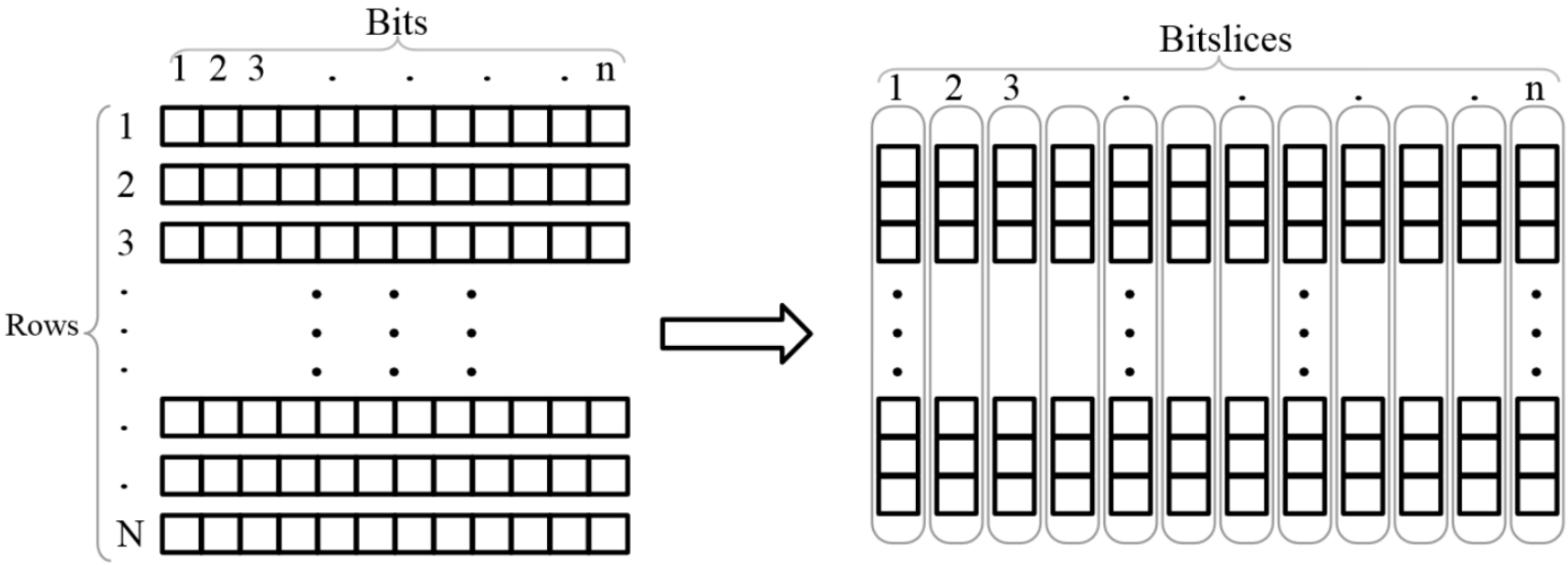
- Deterministic error bounds
- Runs for a number of iterations that can be specified
- The accuracy of the results is increased with each iteration
- There is a bounded number of iterations required to achieve an exact result

Bitwise DAQ

The *bitwise* DAQ scheme iterates over the bits of all values.

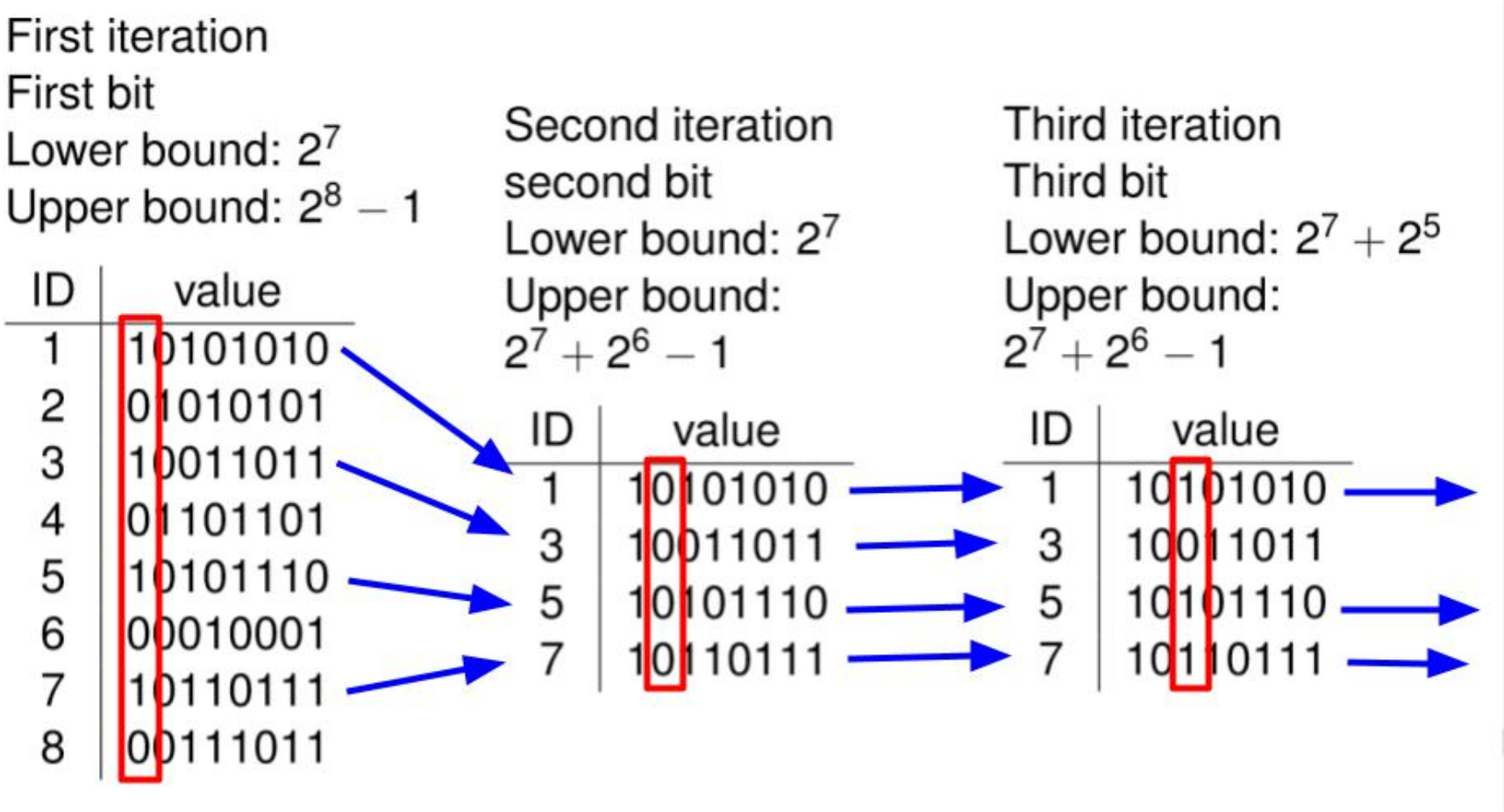
On the first iteration it only looks at the most significant bit, on the second only the second most significant bit, and so on

Bitslice index

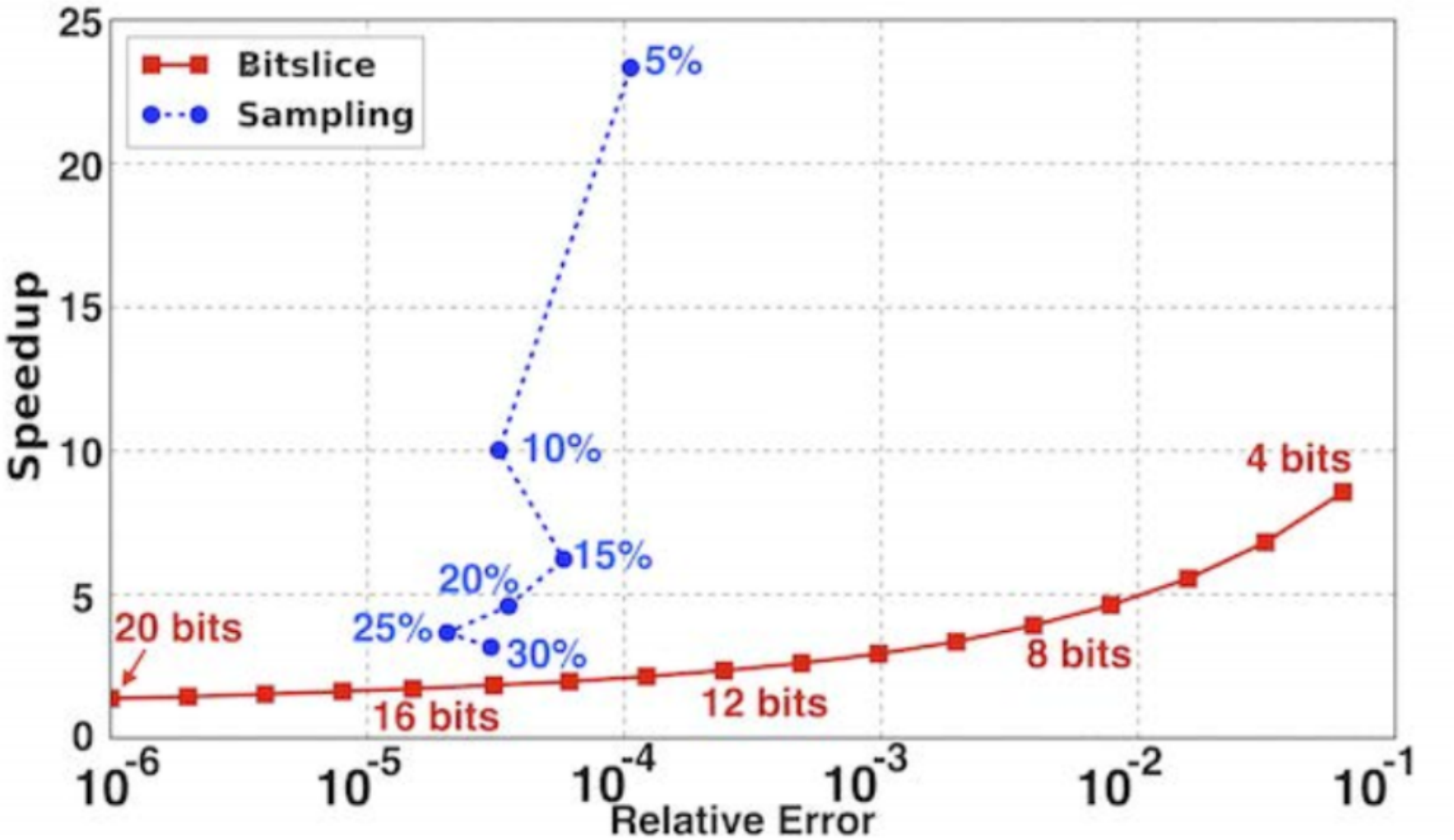


Source - DAQ: A New Paradigm for Approximate Query Processing

Example: approximating the maximum of a 8-bit integer field with bitwise DAQ.

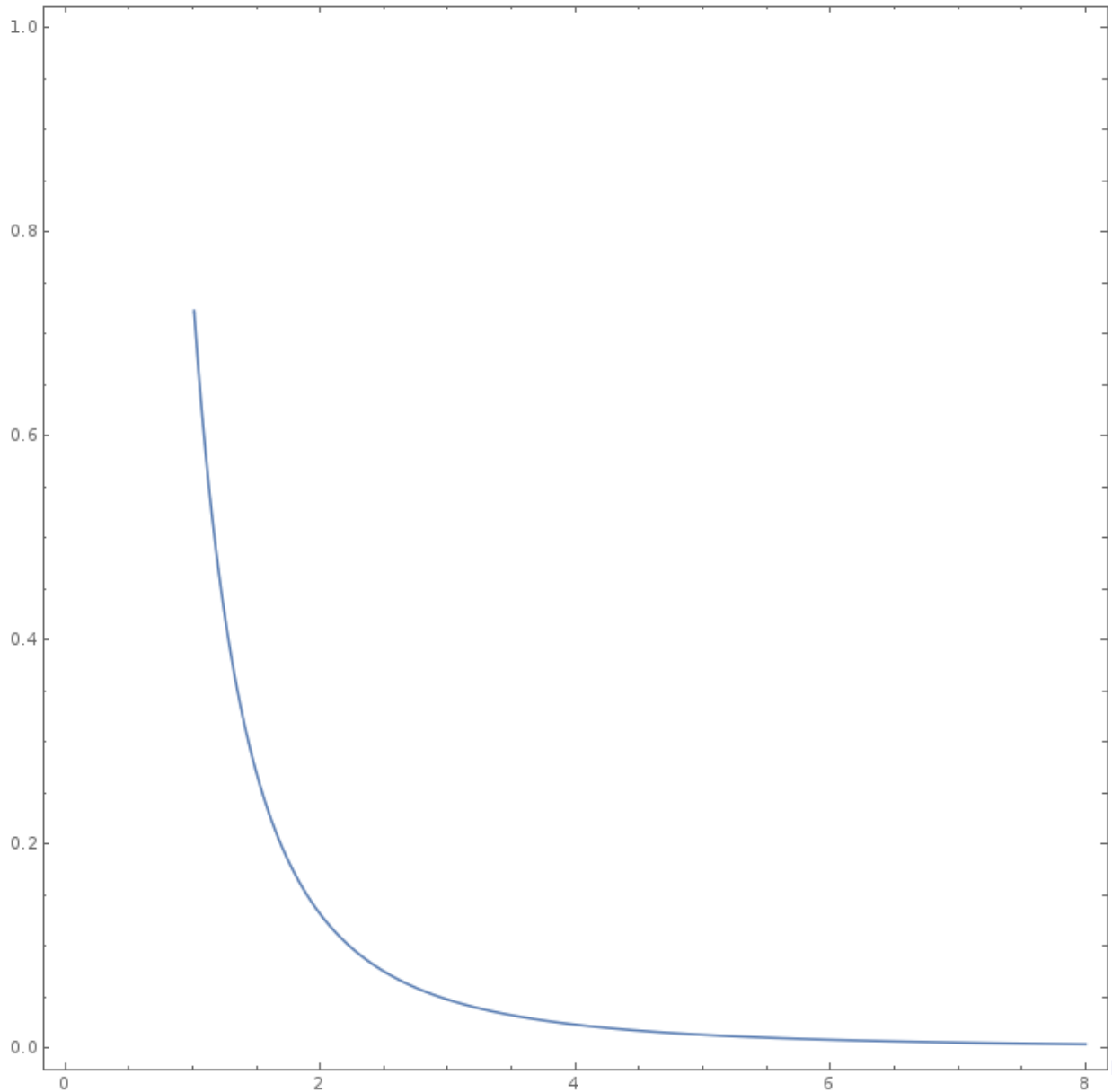


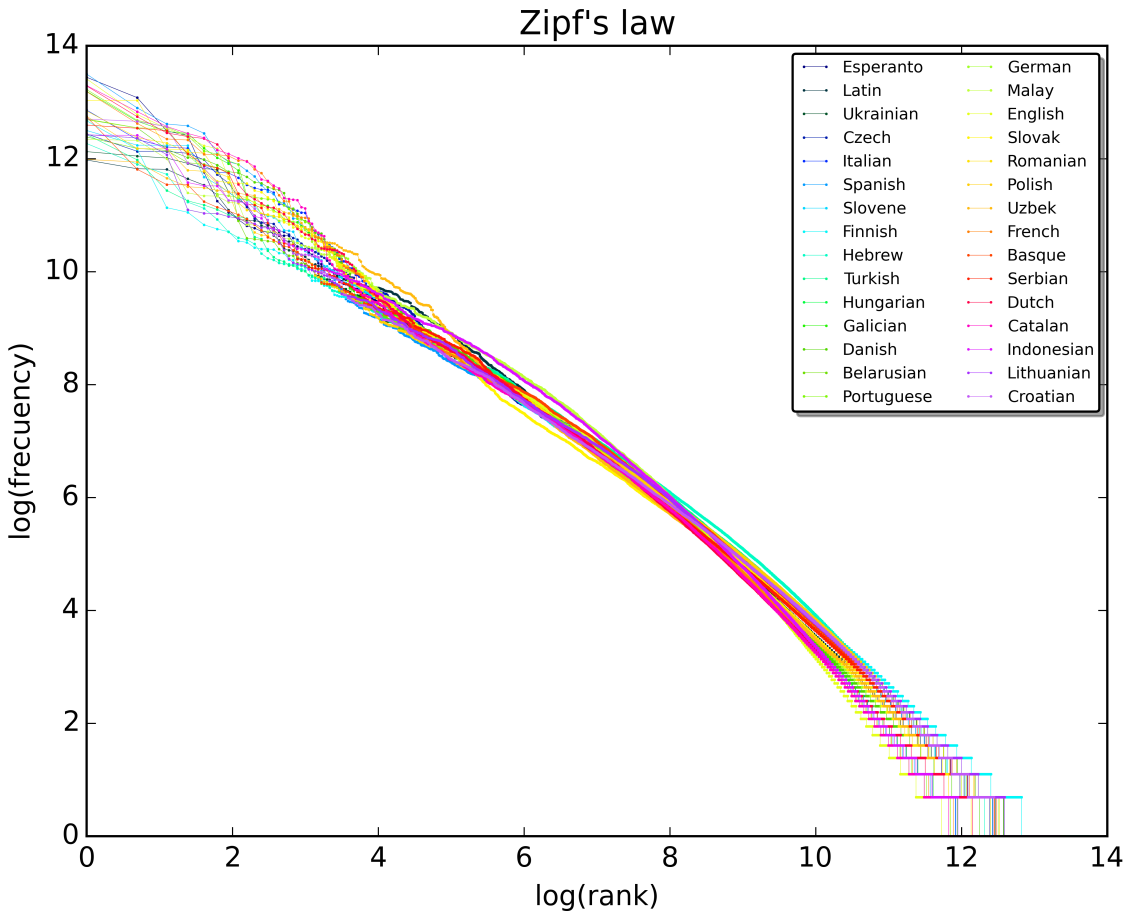
SAQ vs DAQ with Average on Uniform Distributions



Source - DAQ: A New Paradigm for Approximate Query Processing

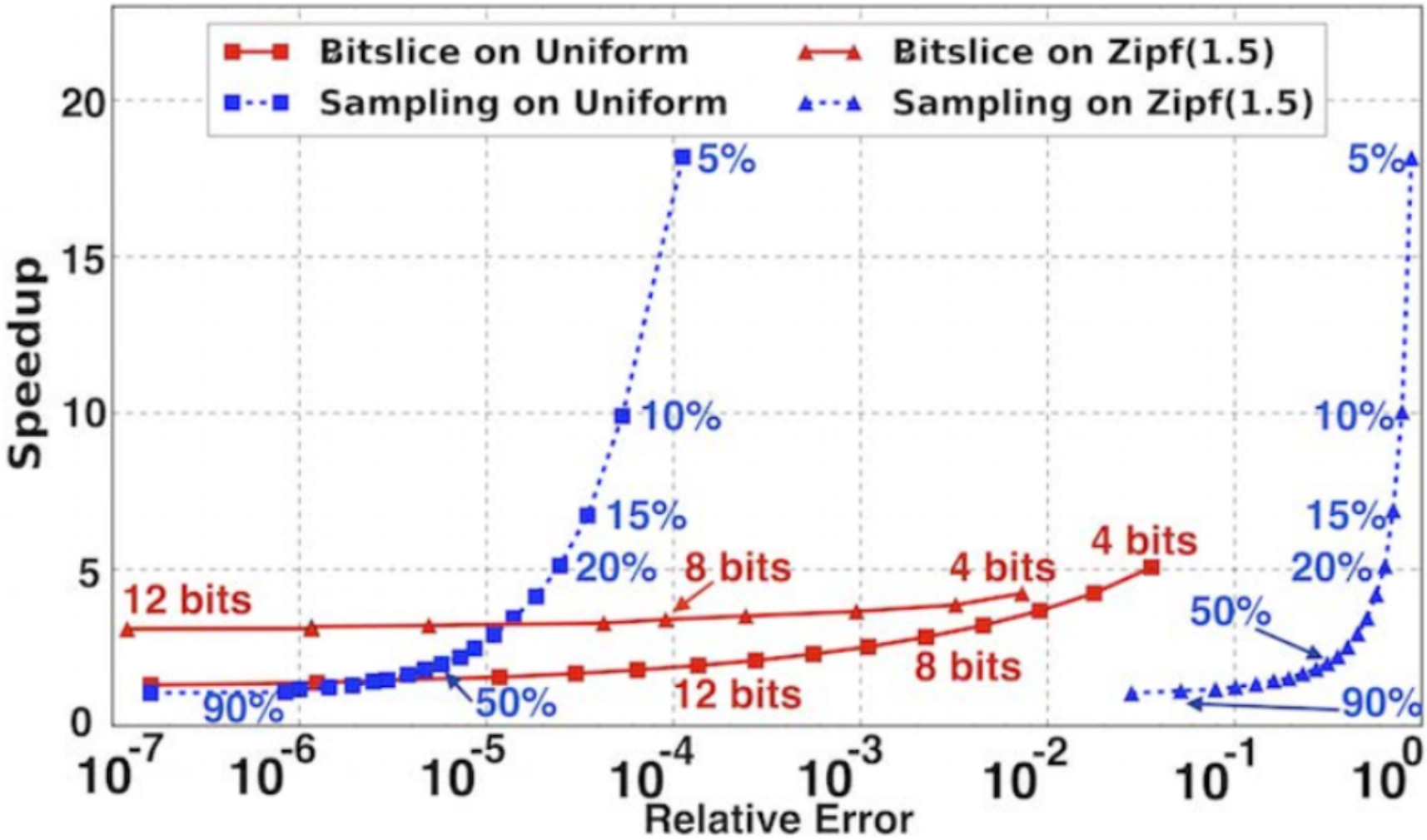
Zipfian Distribution





Source - Wikipedia: Zipf's Law

SAQ vs Bitwise DAQ with top100



Source - DAQ: A New Paradigm for Approximate Query Processing

Outline

Conclusion

- Scale independence and bounded evaluability show promising results
- SAQ is suitable for some queries
- DAQ shows some promise, and is good at a lot of things SAQ is bad at

Acknowledgements

- Elena Machkasova, my senior seminar professor and advisor
- Peter Dolan, for helping me understand basic concepts
- Jacob Opdahl, my alumni reviewer

References

- Y. Cao, W. Fan, Y. Wang, T. Yuan, Y. Li, and L. Y.Chen. Beas: Bounded evaluation of SQL queries. In Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems
- W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems
- N. Potti and J. M. Patel. Daq: A new paradigm for approximate query processing. Proc. VLDB Endow.