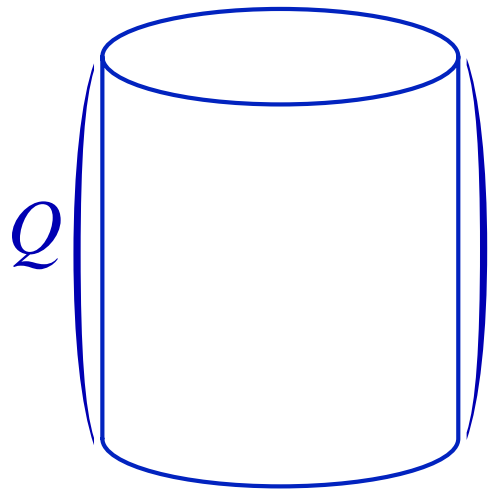
Scale Independence: Using Small Data to Answer Queries on Big Data

Floris Geerts

University of Antwerp

Query answering on big data

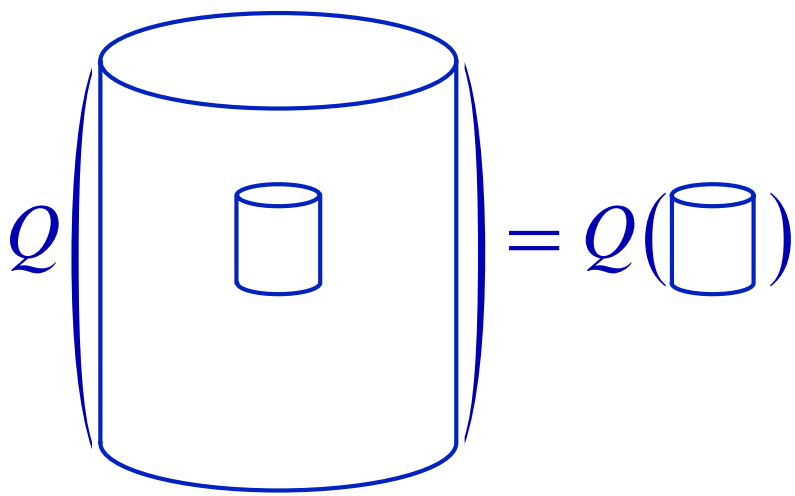
- Queries can be **slow** on big data due the **size of the data**:



- Current database technology tells us how to **quickly** answer queries on **"normal"-sized** data.

Query answering on big data

- It would be great if we can answer Q on a **big database** using a **small database** inside it!
- One could then rely on **existing database technology** to answer queries on **big data**.



Scale independent queries

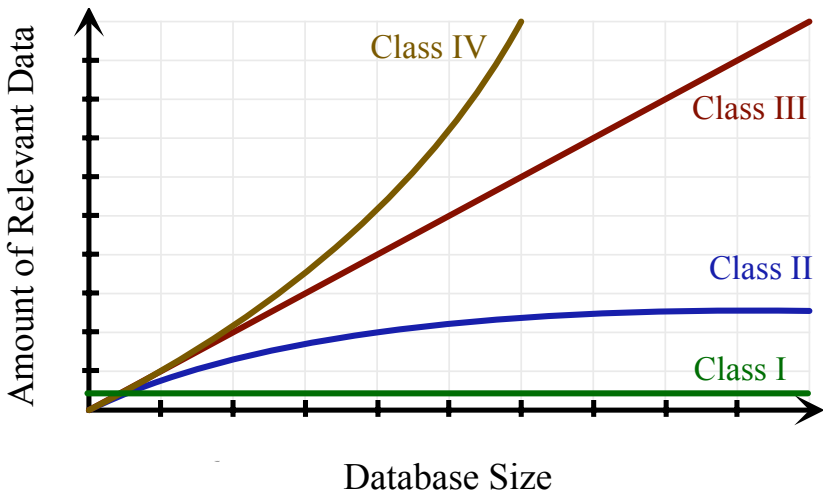
- This idea has been pursued before...

  "Scale independent queries that satisfy their performance objectives on small data sizes will continue to meet those objectives as the database size grows, ..."

  - M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh. Scads: Scale-independent storage for social computing applications. In CIDR, 2009.
  - M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. In VLDB, 2011.
  - M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In SIGMOD, 2013.

Query scaling classes

- Armbrust et al. distinguish between different query classes.
- Depending on **how much data is needed to answer** them:



- Class I: constant amount of data (key value);
- Class II: bounded amount of data (5000 facebook friends);
- Class III: (sub-)linear amount of data;
- Class IV: superlinear (cartesian product).

- **Class I & II queries** are the focus of this talk.

Questions:

1. Which queries are scale independent?
2. How to define this notion?

Scale independent queries

A query Q is scale independent in a database D if

- $Q(D) = Q(D_Q)$ for some **part** $D_Q \subseteq D$; and
- the size $|D_Q|$ of D_Q is **independent** of the size $|D|$ of D.

A query Q is scale independent if it is scale independent in **all databases.**

To answer scale independent queries one only needs a **bounded amount of data**.

- Wenfei Fan, F, Leonid Libkin. On scale independence for querying big data. In PODS 2014.

Checking scale independence

Not surprisingly, for **FO queries**:

- Testing scale independence is undecidable.
- The class of scale independent FO queries is not even recursively enumerable.

Scale independence can also be **uninteresting**.

For **(U)CQ queries**:

- Only trivial CQ queries can be scale independent, e.g., queries that output a constant tuple on all databases.
- This is due to monotonicity.

Undecidable or uninteresting: End of story...

Thank you for your attention.

We need some stronger assumptions on the data

PIQL: Performance-Insightful Query Language

Armbrust et al. propose an **extension of SQL** that allows to express

- **relationship cardinalities**;
- **result size** requirements

+ compiler that **bounds** the number of operations performed by query.

Schema:

```
facebook(member_id,friend_id)
```

Constraint:

```
facebook[member_id -> friend_id, 5000]
```

Query:

```
SELECT friend_id
FROM facebook
WHERE member_id=Trump
```

$\Rightarrow$ Only requires **fetching** at most 5000 tuples.
(Probably less for Trump)

- M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. In VLDB, 2011.

PIQL: Constraints

Constraints are **crucial** in PIQL:

Relation `facebook(id1, id2)`
+ cardinality constraint `facebook[id1 -> id2, 5000]` (aka Facebook constraint)

Relation `person(id, name, city)`
+ key constraint `person[id -> {person, city},1]`

These constraints:

- **bound** the amount of data; and
- specify **access patterns** (how to access the data)

} access constraints

Access constraints

$$\left\{\begin{array}{l} \texttt{person[id -> \{person, city\},1]} \\ \texttt{facebook[id1 -> id2, 5000]} \end{array}\right\}$$

- **Available indexes**:
  - Given an `id`-value one can efficiently fetch the corresponding `name` and `city` values from the `person` table.
  - Similarly for the `facebook` relation.
- **Cardinality constraints**:
  - At most one tuple in `person` table for each `id`-value (key);
  - At most 5000 friends for each member of Facebook.

Can we **make queries scale independent** by using such constraints?

Example: Query evaluation using access constraints

Consider query

$$Q(\text{Trump}, \text{name}) = \exists \text{id}\, \texttt{facebook}(\text{Trump}, \text{id}) \wedge \texttt{person}(\text{id}, \text{name}, \text{NYC})$$

Execution plan:

1. **Fetch all** id-**values** associated with Trump using **index** `facebook[id1 -> id2]`

   $\Rightarrow$ at most 5000 tuples.
2. **For each** of the fetched id-values, **fetch** unique tuple in `person` using **index** `person[id-> { name, city}]`

   $\Rightarrow$ at most 1 tuple per id-value.
3. Return **all fetched** name-values for persons living in NYC.

   $\Rightarrow$ at most 10000 tuples in total.

If such execution plan exists, then we say that a query is boundedly evaluable.

Boundedly evaluable queries

We next define what boundedly evaluable queries are.

Databases that satisfy access constraints

- A database D conforms to a set $\mathcal{A}$ of access constraints, if it **satisfies** these constraints.
  - For each $R(X \to Y, N)$ in $\mathcal{A}$, we have that $|\pi_Y(\sigma_{X=\bar{a}}(D))| \leqslant N$ for any constant tuple $\bar{a}$.

- When looking at query equivalence, we mean $\mathcal{A}$-equivalence, i.e., equivalence on **all databases that conform to the access constraints in $\mathcal{A}$.**

Boundedly evaluable queries

- A query Q is boundedly evaluable relative to a set $\mathcal{A}$ of access constraints if
  - there exists a bounded query plan ξ_Q such that
  - for all databases D conform to $\mathcal{A}$

$$Q(D) = \xi_Q(D).$$

Bounded query plans ensure that only **a bounded amount of data is fetched** from the underlying data.

- Wenfei Fan, F, Leonid Libkin: On scale independence for querying big data. PODS 2014.
- Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data, PODS, 2015.

Bounded query plans (bqplan)

- Starting from basic fetch operators, pipelining using constraints in $\mathcal{A}$:

$$\text{bqplan} = \textbf{fetch}(X = \bar{a}, R, Y)$$
$$\text{bqplan} = \textbf{fetch}(X \in \text{bqplan}, R, Y)$$

- **SPJ-plan**: Allowing projection, selection, renaming and product:

$$\text{bqplan} = \pi_X(\text{bqplan})$$
$$\text{bqplan} = \sigma_C(\text{bqplan})$$
$$\text{bqplan} = \rho_{A/B}(\text{bqplan})$$
$$\text{bqplan} = \text{bqplan} \times \text{bqplan}$$

- **SPJU-plan**: Further allowing union:

$$\text{bqplan} = \text{bqplan} \cup \text{bqplan}$$

- **RA-plan**: Also allowing difference:

$$\text{bqplan} = \text{bqplan} \setminus \text{bqplan}.$$

Example: Bounded query plan

Consider query

$$Q(\text{Trump}, \text{name}) = \exists \text{id}\, \texttt{facebook}(\text{Trump}, \text{id}) \wedge \texttt{person}(\text{id}, \text{name}, \text{NYC})$$

Bounded query plan:

1. $\text{bqplan}_1(\text{id}_1, \text{id}_2) = \textbf{fetch}(\text{id}_1 = \text{Trump}, \texttt{facebook}, \text{id}_2)$
2. $\text{bqplan}_2(\text{id}_2) = \pi_{\text{id}_2}(\text{bqplan}_1)$
3. $\text{bqplan}_3(\text{id}, \text{name}, \text{city}) = \textbf{fetch}(\text{id} \in \text{bqplan}_2, \texttt{person}, (\text{name}, \text{city}))$
4. $\text{bqplan}_4(\text{id}, \text{name}, \text{city}) = \sigma_{\text{city}=\text{NYC}}(\text{bqplan}_3)$
5. $\text{bqplan}(\text{id}_1, \text{name}) = \pi_{\text{id}_1}(\text{bqplan}_1) \times \pi_{\text{name}}(\text{bqplan}_4)$.

On databases D **conform** to $\mathcal{A}$, this query plan **correctly evaluates** Q and fetches a **bounded amount of data**.

Question:

- What do we gain by this definition?

Bounded evaluation makes scale independence a bit more interesting

Although it is still **undecidable** to decide whether an FO query is boundedly evaluable...

...the **presence** of access constraints allow to identify **interesting classes of queries** that are boundedly evaluable.

And this for CQ, UCQ, and FO queries.

Question:

- Which queries are boundedly evaluable?
- Déjà vu?

Querying under access patterns...

Looks similar to the work on **querying under limited access patterns** by Li, Chang, Deutsch, Nash, Ludäscher and others.

Consider query

$$Q(\text{Trump}, \text{name}) = \exists \text{id}\, \texttt{friend}(\text{Trump}^i, \text{id}^o) \wedge \texttt{person}(\text{id}^i, \text{name}^o, \text{NYC}^o)$$

and **access patterns** $\texttt{friend}(\text{id}^i, \text{id}^o)$ and $\texttt{person}(\text{id}^i, \text{name}^o, \text{city}^o)$ indicating **in-and output positions**.

Can it be answered using a **valid access pattern sequence** (query plan)?

- Chen Li, Edward Y. Chang: On Answering Queries in the Presence of Limited Access Patterns. ICDT 2001.
- Alin Deutsch, Bertram Ludäscher, Alan Nash: Rewriting queries using views with access patterns under integrity constraints. Theor. Comput. Sci, 2007.
- Alan Nash, Bertram Ludäscher: Processing First-Order Queries under Limited Access Patterns. PODS 2004.
- ...

Querying under access patterns

Syntactic condition:

A query is orderable when one can parse it from left to right using access patterns.

- Orderable queries have a **linear executing plan** using access patterns.

Semantic condition:

Query is executable/stable if it is **equivalent** to an orderable query.

- Characterizations and complexity results for executable queries are known.

Most of this work considers CQ and UCQ, but also fragments of FO.

Querying under access patterns vs access constraints

However,

- Access patterns cover **all attributes** in relations; access constraints are more **flexible**; and
- **No cardinality constraints** are embedded in access patterns.
- **Standard equivalence** vs. $\mathcal{A}$**-equivalence.**
- Query plans are **not bounded**.

Querying under access patterns vs access constraints

Nevertheless, it serves as **inspiration:**

Access patterns		Access constraints
Executable/stable queries	$\mapsto$	Boudedly evaluable queries
Orderable queries	$\mapsto$	??

We next generalize the notion of orderable queries in the context of **access constraints**.

Covered queries

1. Define a **syntactic fragment** of conjunctive queries:
   $\Rightarrow$ covered queries.
2. Covered queries are boundedly evaluable (SPJ-plan).
3. Every boundedly evaluable CQ is $\mathcal{A}$-equivalent to a covered CQ.

Access patterns		Access constraints
Executable/stable queries	$\mapsto$	Boudedly evaluable queries
Orderable queries	$\mapsto$	covered queries

- Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data, PODS, 2015.
- Yang Cao, Wenfei Fan, Tianyu Wo, Wenyuan Yu. Bounded Conjunctive Queries. PVLDB, 2014.

Covered conjunctive queries

Intuitively, a conjunctive query is covered if

(i) all its relevant variables are **bounded** by access constraints

(ii) all its relations are properly **indexed**.

Precise definition uses **deduction rules** that propagate information on bounds and indexes **based on the structure** of the query.

It is in **PTIME** to check whether a CQ is covered.

If covered, the successful deductive proof **generates** a bounded query plan. (Hence, covered queries are indeed boundedly evaluable.)

Covered CQs: Deduction rules

(i) **Bounding the free variables** using access constraints.

Deduction rules $\mathcal{I}_{Bnd}$:

(Reflexivity)
If $\bar{x}' \subseteq \bar{x}$ then $\bar{x} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{x}', 1)$
(Actualization)
If $R(X \rightarrow Y, N) \in \mathcal{A}$ then $\bar{x} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{y}, N)$
(Augmentation)
If $\bar{x} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{y}, N)$ then $\bar{x} \cup \bar{z} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{y} \cup \bar{z}, N)$
(Transitivity)
If $\bar{x} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{y}_1, N_1)$ and $\bar{y}_1 \rightarrow_{\mathcal{I}_{Bnd}} (\bar{z}, N_2)$ then $\bar{x} \rightarrow_{\mathcal{I}_{Bnd}} (\bar{z}, N_1 \cdot N_2)$

A conjunctive query $Q(\bar{x})$ is bounded if for each $x \in \bar{x}$

$$\Sigma_Q \rightarrow_{\mathcal{I}_{Bnd}} (x, N_x)$$

for some $N_x \in \mathbb{N}$, where Σ_Q are the variables in Q bound to a constant.

Example: Query with "bounded" variables

Consider query

$$Q(\mathrm{id}_1, \mathrm{name}) = \exists \mathrm{id}, \mathrm{city}\, \mathtt{facebook}(\mathrm{id}_1, \mathrm{id}) \wedge \mathtt{person}(\mathrm{id}', \mathrm{name}, \mathrm{city}) \wedge \mathrm{id}_1 = \mathit{Trump} \wedge \mathrm{city} = \mathit{NYC} \wedge \mathrm{id} = \mathrm{id}'$$

Is the variable "name" bounded?

1. $\Sigma_Q = \{\mathrm{id}_1, \mathrm{city}\}$
2. $\Sigma_Q \rightarrow_{\mathcal{I}_{Bnd}} (\mathrm{id}, 5000)$ (Actualization)
3. $\mathrm{id}' \rightarrow_{\mathcal{I}_{Bnd}} (\mathrm{name}, \mathrm{city}, 1)$ (Actualization)
4. $\Sigma_Q \rightarrow_{\mathcal{I}_{Bnd}} (\mathrm{name}, \mathrm{city}, 5000)$ (Transitivity)

Similarly for variable id_1. Hence,

$$\Sigma_Q \rightarrow_{\mathcal{I}_{Bnd}} \{(\mathrm{id}_1, 1), (\mathrm{name}, 5000)\}.$$

It takes $O(|Q|(|\mathcal{A}| + |Q|))$ time to check whether a CQ query is bounded.

Example: Boundedness alone does not suffice

Consider query:

$$Q(\text{user}, \text{photo}, \text{time}, \text{location}) = \texttt{Instagram}(\text{user}, \text{photo}, \text{time}, \text{location}) \wedge \text{user} = \text{Trump} \wedge \text{location} = \text{Bordeaux}.$$

Access constraints:

$\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{photo}, N)]$
$\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{time}, N')]$

Then, using the deduction rules one can show that **all variables in Q are bounded.**

Nevertheless, Q is **not boundedly evaluable.**

Example: Boundedness alone does not suffice

Access constraints

$\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{photo}, N)]$ $\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{time}, M)]$

Indexes can only fetch parts of the relation and **full relation cannot be recovered** (lossy decomposition):

Trump	*photo*1	*time*1	*Bordeaux*
Trump	*photo*2	*time*2	*Bordeaux*
Trump	*photo*3	*time*3	*Bordeaux*

$\neq$

Trump	*photo*1	*Bordeaux*
Trump	*photo*2	*Bordeaux*
Trump	*photo*3	*Bordeaux*

$\times$

Trump	*time*1	*Bordeaux*
Trump	*time*2	*Bordeaux*
Trump	*time*3	*Bordeaux*

Covered CQs: Revised deduction rules

Ensure that access constraints **suffice to correctly check existence of tuples in base relations**.

$\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{photo}, N)]$ $\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{time}, N')]$

Additional access constraint is needed, e.g.,

$$\texttt{Instagram}[(\text{photo}, \text{time}) \to (\text{user}, \text{photo}, \text{time}, \text{location}, N'')]$$

A **refinement** of the deduction rules $\mathcal{I}_{Bnd}$ can be defined such that when **all relevant variables are bounded and indexed**, then the query is boundedly evaluable.

- Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data, PODS, 2015.
- Yang Cao, Wenfei Fan, Tianyu Wo, Wenyuan Yu. Bounded Conjunctive Queries. PVLDB, 2014.

Automatic bounded query plan generation

Access constraints:

$\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{photo}, N)]$ $\texttt{Instagram}[(\text{user}, \text{location}) \to (\text{time}, N')]$
$\texttt{Instagram}[(\text{photo}, \text{time}) \to (\text{user}, \text{photo}, \text{time}, \text{location}, N)]$

Deductive proof **automatically gives bounded query plan**:

1. $\text{bqplan}_1(\text{photo}) = \pi_{\text{photo}}(\textbf{fetch}((\text{Trump}, \text{Bordeaux}), \texttt{instagram}, \text{photo}))$
2. $\text{bqplan}_2(\text{time}) = \pi_{\text{time}}(\textbf{fetch}((\text{Trump}, \text{Bordeaux}), \texttt{instagram}, \text{time}))$
3. $\text{bqplan}_3(\text{photo}, \text{time}) = \text{bqplan}_1(\text{photo}) \times \text{bqplan}_2(\text{time})$
4. $\text{bqplan}_4(\text{user}, \text{photo}, \text{time}, \text{location}) =$
   $\textbf{fetch}((\text{photo}, \text{time}) \in \text{bqplan}_3, \texttt{instagram}, (\text{user}, \text{photo}, \text{time}, \text{location}))$
5. $\text{bqplan}_5(\text{user}, \text{photo}, \text{time}, \text{location}) = \sigma_{\text{name}=\text{Trump} \wedge \text{location}=\text{Bordeaux}}(\text{bqplan}_4)$.

Question:

- How **good** are these query plans **in practice**?

Covered CQs: Experiments

Data:

- UK traffic accident data (19 tables, 113 attributes, 89.7 million tuples, 21.4GB)
- Ministry of Transport Test data (1 table, 36 attributes, 55 million tuples, 16.2GB)
- TPCH (restricted to 8 tables, varying sizes up to 32GB)

Access constraints:

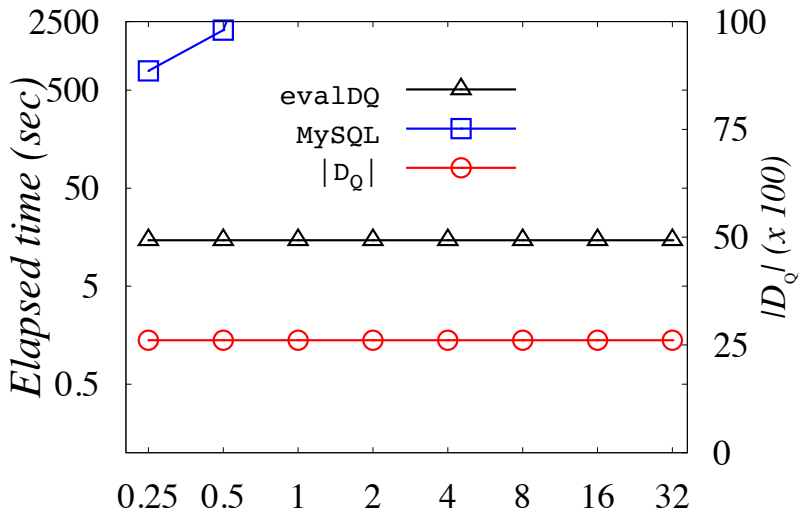
- UK traffic accident data: *84 constraints* (e.g., `[date -> (aid, 610)]`
- Ministry of Transport Test data: *27 constraints*
- TPCH: *61 constraints*

Queries: 15 queries on each dataset (varying selection conditions and # joins).

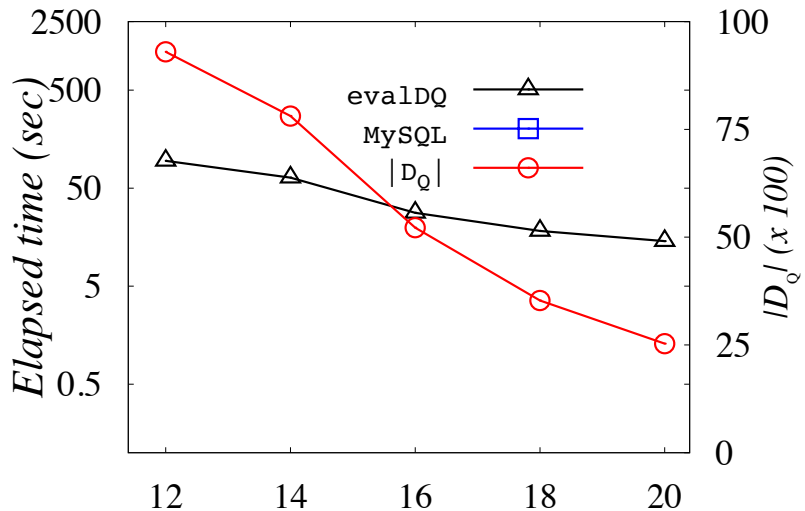
Covered CQs: Experiments - TPCH

- 70% of queries turned out to be boundedly evaluable (when all access constraints were "on")

Comparison between **generated bounded query plan** vs **mysql query plan**:



Average time vs $|D|$



Average time vs $|\mathcal{A}|$

- Yang Cao, Wenfei Fan, Tianyu Wo, Wenyuan Yu. Bounded Conjunctive Queries. PVLDB, 2014.

Semantic characterization?

Being covered is a syntactic condition, i.e., it depends on how the query is written.

We also want a semantic characterization:

Suppose that Q is not covered. Is Q $\mathcal{A}$-equivalent to a query Q' that is covered?

Clearly, such queries can also be evaluated in scale independent way:

- Simply execute the bounded query plan for Q'.

Decision algorithm

Given CQ Q, is Q $\mathcal{A}$-equivalent to a covered CQ?

1. **Decompose** $Q \equiv_{\mathcal{A}} Q_1 \cup \cdots \cup Q_k$.
   - Each $Q_i \models \mathcal{A}$ and no redundant Q_i's.
2. **Compute** the infimum query $\mathbf{inf}_{\mathcal{A}}(Q)$ of $\{Q_1, \ldots, Q_k\}$.
   - For any other CQ Q' such that $Q_i \subseteq Q'$ for all i, we have that $\mathbf{inf}_{\mathcal{A}}(Q) \sqsubseteq Q'$.
3. **Construct** $\mathcal{A}$-expansion $\mathbf{exp}_{\mathcal{A}}(Q)$ of $\mathbf{inf}_{\mathcal{A}}(Q)$.
   - All possible "covered" atoms embedded in $\mathbf{inf}_{\mathcal{A}}(Q)$ are added.
4. **Identify** maximal covered subquery Q_c in $\mathbf{exp}_{\mathcal{A}}(Q)$.

Theorem

A conjunctive query Q is $\mathcal{A}$-equivalent to a covered CQ Q if and only if Q is $\mathcal{A}$-equivalent to the covered CQ Q_c.

Complexity

- The characterisation implies a co2NEXPTIME upper bound:
  - exponential number of base queries Q_i;
  - infimum query is exponential in the number (and size) of base queries.
- Lower bound is open.

Beyond CQ

What can we say about other query languages?

Boundedly evaluable queries: UCQ

For **unions of conjunctive queries** (UCQ)

- Bounded query plans may use union (SPJU-plan).
- Notion of covered UCQ can be defined.
- Every boundedly evaluable UCQ is $\mathcal{A}$-equivalent to a covered UCQ.
- Without impact on complexity.

Similarly for **conjunctive queries with (nested) unions.**

- Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data, PODS, 2015.

Boundedly evaluable queries: First-order logic

A notion of covered FO queries has recently been proposed:

1. **Convert** FO query Q to **relational algebra** expression e_Q.
2. Require in the **query tree** T_{e_Q} of e_Q that:
   - Every **max conjunctive subtree is covered.**

(I.e., difference is pushed to top levels on unions of covered sub-queries)

Theorem

Every covered FO query is boundedly evaluable (RA-plan) and every boundedly evaluable FO query is $\mathcal{A}$-equivalent to a covered FO query.

It takes **PTIME** to check whether an FO query is covered.

- Yang Cao, Wenfei Fan: An Effective Syntax for Bounded Relational Queries. SIGMOD 2016.

Boundedly evaluable queries: First-order logic

A RA bounded query plan can be **generated** for covered FO queries.

Underlying idea:

1. **Encode** Q and $\mathcal{A}$ as a **hypergraph**;
2. A **hyperpath** corresponds to a **bounded query plan**.

Algorithm: Find a hyperpath.

Experiments show that these query plans also **outperform** those used by mysql.

Generating plans from proofs

In recent work by Benedikt et al., the following approach is followed:

1. Isolate a **semantic property** that any input query Q must have with **respect to the class of target plans** in order to have an **equivalent plan of the desired type**.
2. **Express** this property as a **proof goal**: a statement that formula ϕ_2 follows from ϕ_1.
3. **Search for a proof** of the entailment, within a given proof system.
4. From the **proof, extract a plan**.

- Michael Benedikt, Balder ten Cate, Efthymia Tsamoura: Generating Plans from Proofs. ACM Trans. Database Systems, 2016. Based on PODS 2014 paper.

Semantic property: Access determinacy

In the context of access patterns (not access constraints!):

A query Q is said to access determined if

- for any D and D' that have the same **accessible part**

$$\mathbf{AccPart}(D) = \mathbf{AccPart}(D')$$

- it holds that $\mathbf{Q}(\mathbf{D}) = \mathbf{Q}(\mathbf{D'})$.

Intuitively, $\mathbf{AccPart}(D)$ are all values that can be accessed from D.

Clearly, Q cannot be answered using access patterns if it is not access-determined.

Access determinacy: Entailment

An FO query Q is **access determined** if and only if

$$Q \wedge \text{Access}^{+} \models Q_{acc}$$

where Q_{acc} is the **inferred accessible version** of Q

- obtained by replacing each R in Q by its accessible part

and **Access**$^{+}$ is an axiomatization of accessibility:

- rules that tell what is accessible and what not, based on access patterns.

Semantic property: Access monotonic determinacy

A query Q is said to access monotonic determined if

- for any D and D' that have **contained accessible parts**

$$\mathbf{AccPart}(D) \subseteq \mathbf{AccPart}(D')$$

- it holds that $\mathbf{Q(D)} \subseteq \mathbf{Q(D')}$.

Access monotonic determinacy: Entailment

A CQ query Q is **access monotonic determined** if and only if

$$Q \wedge \text{Access} \models Q_{acc}$$

where Q_{acc} is the inferred accessible version of Q and **Access** is an axiomatization of accessibility:

- rules that tell what is accessible, based on access patterns.

Access determinacy: Plans from proofs

Nice property: Chase proofs witnessing

$$Q \wedge \text{Access} \models Q_{acc}$$

or

$$Q \wedge \text{Access}^{+} \models Q_{acc}$$

result in **SPJ and RA-plans**, for CQ and FO queries, respectively.

Furthermore, **cost functions** can be incorporated to find **cost optimal** proofs (plans).

Future work: Bounded access (monotonic) determinacy?

The following seems a natural thing to try:

1. Define a notion of **bounded** access (monotonic) determinacy.
2. Consider **access constraints** instead of access patterns.
3. Taking into account $\mathcal{A}$-**equivalence**.
4. **Extract** bounded query plans from **proofs**.

TODO...

Recap

- Successfully identified class of **covered queries** that are **boundedly evaluable**.
- **Every boundedly evaluable query** is $\mathcal{A}$-equivalent to **covered** one.
- Definition of covered queries "implies" bounded query plan **generation procedure.**
- Generated query plans **work well in practice.**

Other issues

- Scale independent **query approximation.**
- **Incremental** scale independence.
- Scale independence using **views**.

Scale independent query approximation

Given a query Q that is **not** boundedly evaluable.

Find **two boundedly evaluable** queries Q_ℓ (**lower envelope**) and Q_u (**upper envelope**) such that

$$Q_\ell \sqsubseteq_{\mathcal{A}} Q \sqsubseteq_{\mathcal{A}} Q_u.$$

and Q_ℓ is **maximal** and Q_u is **minimal** wrt $\mathcal{A}$-containment.

- Solved for **CQ**, when Q_u and Q_ℓ are assumed to be **covered** CQ queries.
- Characterization and complexity results are known.
- Full treatment is required....

- Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data, PODS, 2015.

Query approximation: Example

Consider

$$Q(x) = \exists y, z, w\,(\mathtt{R}(w, x) \wedge \mathtt{R}(y, w) \wedge \mathtt{R}(x, z) \wedge w = 1)$$

and access constraint

$$\mathtt{R}(A \to B, N).$$

Then Q is **not boundedly evaluable**.

We can sandwich Q between two boundedly evaluable queries:

$$Q_\ell(x) = \exists y, z\,(\mathtt{R}(1, x) \wedge \mathtt{R}(y, 1) \wedge \mathtt{R}(x, z) \wedge \mathtt{R}(x, y))$$

and

$$Q_u(x) = \exists y, z\,(\mathtt{R}(1, x) \wedge \mathtt{R}(x, z)).$$

Furthermore, $|Q(D) \setminus Q_\ell(D)| \leqslant N$ and $|Q_u(D) \setminus Q(D)| \leqslant N$.

Such envelopes, if they exist, can be obtained by **relaxing and generalizing** the input query.

Incremental scale independence

$\mathbf{\Delta}D = (\Delta D, \nabla D)$: List of tuples ΔD to be **inserted** into D and a list ∇D of tuples to be **deleted**.

$\mathbf{\Delta}Q = (\Delta Q, \nabla Q)$: queries such that

$$Q((D \setminus \nabla D) \cup \Delta D)) = (Q(D) - \nabla Q(\mathbf{\Delta}D, D)) \cup \Delta Q(\mathbf{\Delta}D, D)$$

Then, Q is incrementally boundedly evaluable iff

- ΔQ is boundedly evaluable; and
- ∇Q is boundedly evaluable.

That is, to incrementally answer Q in D in response to $\mathbf{\Delta}D$, we need to access a bounded number of tuples from D

- Wenfei Fan, F, Leonid Libkin: On scale independence for querying big data. PODS 2014.
- M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In SIGMOD, 2013.

Scale independence using views

Enlarge the class of boundedly evaluable queries by using cached views:

- Cached views allow **fast access** $\Rightarrow$ **all view data** can be used.

Extension of bounded query plans:

- Allow to fetch data from views in an **unrestricted way**.

Complication:

- Ensure that views only pass a **bounded** amount of data to indexes (access constraints) on **base relations**.

Complexity results and effective syntax for CQ and FO are established, **assuming a (constant) bound on the size of query plans.**

- Wenfei Fan, F, Leonid Libkin: On scale independence for querying big data. PODS 2014.
- Yang Cao, Wenfei Fan, F., and Ping Lu. Bounded Query Rewriting Using Views. PODS 2016.

Conclusion

- Boundedly evaluable queries are a **nice concept** with interesting links to
  - Safety;
  - Querying using access patterns;
  - Access determinacy and query rewriting.
- Main complications arise from the presence of **cardinality constraints.**
- Experiments show that bounded query plans can **outperform** query plans suggested by optimizer.

Conclusion

- I did not mention **complexity results** for various associated decision problems.
- These can be found here:
  - Yang Cao, Wenfei Fan. An Effective Syntax for Bounded Relational Queries. SIGMOD 2016.
  - Yang Cao, Wenfei Fan, F. and Ping Lu. Bounded Query Rewriting Using Views. PODS 2016.
  - Wenfei Fan, F, Yang Cao, and Ting Deng. Querying Big Data by Accessing Small Data. PODS 2015.
  - Yang Cao, Wenfei Fan, Tianyu Wo, Wenyuan Yu. Bounded Conjunctive Queries. PVLDB 2014.
  - Wenfei Fan, F, Leonid Libkin: On scale independence for querying big data. PODS 2014.

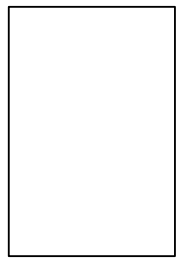
Looking ahead

- **Bounded access determinacy** and generation of bounded query plans from proofs.
- Enlarge class of **covered FO** queries.
- **Index suggestion** to make queries boundedly evaluable.
- Integration with **integrity constraints**.
- Scale-independence in a **distributed/parallel** context.
- Scale-independence on **graph data and query languages.**

Thank you. The End. Questions?







(Thanks to Wenfei Fan, Leonid Libkin, Cao Yang, Ting Deng, Ping Lu)

(and don't forget to send your best work to PODS 2017, 1st deadline June 17, 2016)