Querying Big Data: Theory, Systems and Applications

Wenfei Fan

University of Edinburgh

BDBC, Beihang University
Big data: Through the eyes of computation

- Computer science is the topic about the computation of function $f(x)$

Big data: the data parameter $x$ is horrendously large: PB or EB

Any fundamental challenges introduced by querying big data?

A departure from classical theory and traditional techniques
A Graph Search query (Facebook)

✓ Find me all my friends who have a friend in the US
✓ Database schema: two relations
  • \textit{person}(pid, name, city, country)
  • \textit{friend}(pid1, pid2, common-interest)

\textbf{A simple SPC query (conjunctive query)}

\begin{verbatim}
select  f1.pid2
from    person  p,  friend  f1,  friend  f2
where   f1.pid1 = p0  and  f1.pid2 = f2.pid1  and
         f2.pid2 = p.pid  and  p.country = USA
\end{verbatim}

\textit{How to evaluate the query?}
A conventional query plan

select f1.pid2
from person p, friend f1, friend f2
where f1.pid1 = p0 and f1.pid2 = f2.pid1 and
    f2.pid2 = p.pid and p.country = USA

fetch the friend relation, and find the set F1 of friends of p0

for each tuple t in F1,
  • for each tuple f in the friend relation
    • if t.pid2 = f.pid1
    • then check whether p.country = USA in the corresponding tuple p in person; output t.pid if so

scan the entire relations

a nested loop

assume an index for person

billions of person tuples, trillions of friend links, 300PB of data
How long does it take?

- fetch the *friend* relation, and find the set $F1$ of friends of $p0$
- for each tuple $t$ in $F1$,
  - for each tuple $f$ in the *friend* relation
    - if $t.pid2 = f.pid1$
    - then check whether $p.country = \text{USA}$ in the corresponding tuple $p$ in *person*; output $t.pid$ if so

Facebook: $300\text{PB (10}^{15}\text{B)}$ of data, not 15TB

Samsung, March 2016

Assume (in favor of conventional DBMS)

- *friend* has 15TB of data
- $F1$ has 100 tuples
- an index for *person* relation

- a linear scan: 20 minutes
- 2000 minutes

$2,020 \text{ minutes} = 33 \text{ hours} = 1.4 \text{ days}$
The good, the bad and the ugly

Traditional computational complexity theory of 50 years:

- The good: polynomial time computable (PTIME)
- The bad: NP-hard (intractable)
- The ugly: PSPACE-hard, EXPTIME-hard, undecidable…

What happens when it comes to big data?

Using SSD of 12G/s, a linear scan of a data set $D$ would take

- 0.9 day when $D$ is of 1PB ($10^{15}$B) ($O(n)$ time)
- an SPC query on $D$ of 15TB takes 1.4 days ($O(n^2)$ time)

$O(n^2)$ time is already beyond reach on big data in practice!

Polynomial time queries become intractable on big data!
Tractability revisited for big data

BD-tractable queries: properly contained in $P$ unless $P = NC$

Parallel polylog time for online processing, after PTIME offline one-time preprocessing


Open, like $P = NP$
Parallel query answering?

Can we overcome this by using more computing resources? Using 50,000 processors, assuming linear scalability:

- 2,020 minutes is reduced to 2.4 seconds

That is what those big companies are doing. Does this solve the problem?
Parallelism is not a magic bullet

- Only works for “queries with limited joins”
- Parallel scalability is **NOT** always guaranteed
  - most parallel algorithms published are **NOT** parallel scalable
  - there exist computations that are **NOT** parallel scalable
    - the running time is not reduced no matter how many processors are used
  
- Computational costs + communication cost

- Worse yet, small companies cannot afford to rent 50,000 processors!


Can small companies get the benefits of big data services?

A quest for a new query evaluation paradigm
Overview

1. A resource-constrained paradigm for query processing
   1) Theory: bounded evaluation
2. System: BEAS, Bounded EvAluable Sql
3. Applications
   1) Social media marketing: association rules for graph
   2) Graph functional dependencies: for improving the quality of knowledge bases

Joint work with Yang Cao, Ping Lu, Jingbo Xu, Yanghao Wang, Yinhui Wu and Floris Geerts
A new paradigm of query processing
Querying big data by accessing small data

✓ “For many queries Q, we can compute Q(D) without fetching the entire dataset D”

✓ Question: Can we find, given a query Q and a (possibly big) dataset D, a fraction $D_Q$ of D such that

✓ $Q(D) = Q(D_Q)$, and

✓ its size $|D_Q|$ is independent of $|D|$?

✓ Scale independence

However, few real-life queries are scale independent 😞

PODS 2014 (Fan, Geerts, Libkin)
Bounded evaluability

- Input: A query $Q$ and an access schema $A$
- Question: Can we find, for any (possibly big) dataset $D$ that satisfies $A$, a fraction $D_Q$ of $D$ such that
  - $Q(D)$ is computed by accessing $D_Q$ only, and
  - the time for identifying $D_Q$ is determined by $Q$ and $A$ only?

$|D_Q|$ is independent of the size of $D$

- Fixed cost no matter how big $D$ grows

Making the cost of computing $Q(D)$ independent of $|D|$!

W. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu, querying big data by accessing small data, PODS 2015
A Graph Search query (Facebook)

✓ Find me all my friends who have a friend in the US

select  f1.pid2
from     person  p, friend f1, friend f2
where   f1.pid1 = p0   and   f1.pid2 = f2.pid1   and
        f2.pid2 = p.pid   and  p.country = USA

Access schema: a set of access constraints (cardinality + indices)

✓ friend(pid1 → pid2, 5000)
  • Facebook: 5000 friends per person
  • Index: given pid1, retrieves at most 5,000 pid2
✓ person(pid → (name, country), 1)
  • pid is a key for relation person

How does this help?
Bounded query evaluation

```sql
select f1.pid2
from person p, friend f1, friend f2
where f1.pid1 = p0 and f1.pid2 = f2.pid1 and
      f2.pid2 = p.pid and p.country = USA
```

A bounded query plan

- Fetch at most 5000 fid’s for friends of p0 using the index
- For each fid, fetch her friends’ fids (at most 5000) using the index
- For each fid found, find the corresponding person’s country and name (1 tuple)

In contrast to billions of person tuples, and trillions of friend tuples

Accessing 5000 + 5000 * 5000 * 2 tuples in total
Effectiveness of bounded evaluation

- 77% of SPC queries are boundedly evaluable under 84 simple access constraints
- 67% of relational algebra queries are boundedly evaluable under a few hundred access constraints
- Efficiency: 9 seconds vs. 14 hours of commercial DBMS, and the gap gets larger on bigger data

What is new?

- Identify necessary $D_Q$ needed to answer $Q$, by reasoning about $A$
  - No redundant tuples or attributes
  - Attribute-based indices: retrieve $D_Q$ efficiently

Critical: Decide the existence of $D_Q$ in advance

Experimenting with real-life data

- Y. Cao, W. Fan and T. Wo, Bounded conjunctive queries, VLDB 2014
- Y. Cao and W. Fan, An effective syntax for bounded relational queries, SIGMOD 2016

DBMS: redundancies get inflated rapidly in the presence of joins

Partial tuples vs. entire tuples

friend(pid1, pid2, common_interest)
The bounded evaluability problem

- Input: A relational schema $R$, an access schema $A$, and a query $Q$ in a query language $Q$
- Question: Is $Q$ boundedly evaluable under $A$?

When $Q$ has a bounded query plan under $A$.

No matter how desirable, the problem is hard

- Undecidable for first-order logic FO (the full relational algebra)
- EXPSPACE-hard for SPC (conjunctive queries)

W. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu, Querying big data by accessing small data. PODS 2015

Can we still make practical use of bounded evaluability?
An effective syntax

There exists a class of covered FO queries by an access schema \( A \)

- An FO query \( Q \) is boundedly evaluable under \( A \) if and only if \( Q \) is equivalent to an FO query covered by \( A \)
- All FO queries covered by \( A \) are boundedly evaluable under \( A \)
- It is in \( \text{PTIME} \) to syntactically check whether an FO query is covered by \( A \) in \(|Q|, |A| \) and \(|R| \)

A core sub-class of boundedly evaluable FO queries

- Reduce bounded evaluability to \( \text{PTIME} \) syntactic checking, without losing the expressive power

Over 80% of Boundedly evaluable queries in real-life are covered

Y. Cao and W. Fan, An effective syntax for bounded relational queries, SIGMOD 2016

A syntactic characterization of boundedly evaluable queries
Bounded query rewriting using views

✓ Input: A query $Q$, an access schema $A$, a set $V$ of views, and a natural number $M$

✓ Question: Can we find, for any (possibly big) dataset $D$ that satisfies $A$, a fraction $D_Q$ of $D$ and a query $Q'$ such that

  ✓ $Q(D) = Q(D_Q, V(D))$,
  ✓ the query plan for answering $Q'$ is bounded by $M$, and
  ✓ the time for identifying $D_Q$ is determined by $Q$ and $A$ only?

---

Huawei’s CDR queries:

✓ 90+% are bounded

✓ Improvement: 25 to $10^5$ times

---

Y. Cao, W. Fan, F. Geerts, P. Lu, Bounded query rewriting using views. PODS 2016

Bounded query evaluation and rewriting
Query evaluation with bounded resources

- A resource ratio $\alpha \in [0, 1)$, indicating the amount of data we can afford to access with our available resources
- Access schema $A$, which we can reason about and identify necessary data needed to answer a query
- A set $V(D)$ of small views cached in memory

A new paradigm: querying big data with constrained resources

Given an SQL query $Q$ and a dataset $D$ satisfying $A$

- Decide whether $Q$ is boundedly evaluable under $A$ (using $V$)
- If so, compute exact answer $Q(D)$ by accessing bounded $D$
- Otherwise, compute $Q(D_Q)$ as approximate query answer, with accuracy guarantee

Access at most $D_Q$ in the entire process, $\alpha |D| \geq |D_Q|$
BEAS: a system for \textit{boundedly evaluable} SQL
Resource-constrained query processing

Offline algorithms supported
✓ Discovering access schema A from dataset D of given application
✓ Bounded incremental algorithms for maintaining access constraints in response to updates to D
✓ Selection and maintenance algorithms for views V

Online algorithms supported: given a query Q
✓ Check whether Q is boundedly evaluable
✓ If so, generate a bounded query plan to compute exact answers
✓ Otherwise, generate an approximate plan with accuracy bound
✓ Execute the plans using underlying query engines

Can big companies benefit from BEAS?
Yes, combining bounded evaluation and parallel processing
Resource Bounded Approximate

✓ Input: A resource ratio $\alpha \in [0, 1)$, an accuracy bound $\eta \in (0, 1]$, and an access schema $A$
✓ Question: Develop an algorithm that given any query $Q$ and dataset $D$ that satisfies $A$,

- accesses a fraction $D_Q$ of $D$ such that $|D_Q| \leq \alpha |D|$
- computes $Q(D_Q)$ as approximate answers to $Q(D)$, and
- $\text{accuracy}(Q, D, \alpha) \geq \eta$

Decided by our available resources (time, processors, etc)
Approximate answers

Find a hotel in NYC such that
- it costs around $90 per night
- it is near an art gallery
- it is close to a restaurant that costs about $25 for dinner

```
select hid
from hotel h, POI poi, restaurant r
where h.city = NYC and h.price <= 90 and
    h.zip = poi.zip and h.zip = r.zip and
    poi.type = "art gallery" and t.price <= 25
```

Large: 200 million POI tuples, 0.89 million hotels, and 2.4 million restaurants

Constrained by our available time and resources (e.g., processors)

Can we accurately answer Q by accessing at most 10,000 tuples?
A set $S$ of approximate answers

- **Relevance**: each answer in $S$ is sensible, e.g., hotel $h_5$
  - physical distance: $h_5$ is close to an art museum $p_5$ and to a restaurant $r_3$
  - semantic distance: art museum ($p_5$) is close to art gallery
  - numeric: $h_5$.price ($100$) vs. $90$, and $r_3$.price ($30$) vs. $25$

- **Coverage**: each tuple in the exact answer $Q(D)$ is close to an approximate answer in $S$

Efficient and accurate

Scalable with big data

*the approximate answers are computed by accessing 10000 tuples*
Resource-bounded approximation for SQL

✓ A bi-objective deterministic accuracy metric:
  - **relevance**: every approximate answer in S is close to an exact answer in Q(D)
    - semantic distance, physical distance, etc
  - **coverage**: every exact answer in Q(D) is close to an approximate answer in S

✓ Approximation techniques:
  - A hierarchical representation of data: different “resolutions of the data”, the higher the resolution is, the more accurate
  - Bounded search: guided by Q and A, within budget $\alpha/D$

Performance guarantees for accuracy bound $\eta$, a function of $\alpha$
The novelty of resource-bounded approximation

- Synopsis-based approximation
  - for any database \( D \) compute a synopsis \( D_S \)
  - for all queries \( Q \) posted on \( D \), compute \( Q(D_S) \) as the answer

Aggregate queries only, probabilistic error rate, known workload in advance

Data-driven approximation

- for generic queries, aggregate
- for unpredictable queries, with deterministic accuracy bound

\[ \alpha = 1.5 \times 10^{-6}, \alpha \times 1 \text{PB} (10^{15} \text{B}) = 15 \times 10^9 = 15 \text{GB} \]

- We can making big data of PB size as small as 15GB!

See the demo: Querying big data with constrained resources

W. Fan, X. Wang, and Y. Wu. Querying big graphs within bounded resources, SIGMOD 2014
Application: Graph-pattern association rules
Association rules revised for graphs

✓ Conventional association rules: \( X \Rightarrow Y \)
  - \( X \) and \( Y \): itemsets (attributes of a relation)
  - milk, diaper \( \Rightarrow \) beer

To trump *conventional marketing*
  - 90\% of customers trust their peer (friends, colleagues) recommendations vs. 14\% who trust advertising
  - 60\% of users said that Twitter plays an important roles in their shopping

The need for association rules for graphs
Graph pattern association rules

a) if \( x \) and \( x' \) are friends who live in the same city \( c \),
b) they both like at least 3 French restaurants in \( c \), and
c) if \( x' \) likes a newly opened French restaurant \( y \),
then \( x \) may also like \( y \)

We can advertise restaurant \( y \) to person \( x \)

Association rules defined with graph patterns
Adding logic quantifiers

if all friends of x use Redmi 2A, then x may buy Redmi 2A

Universal quantification: 100%

Existential quantifiers: by default

Universal quantification: 100%
Counting quantifiers and FO operators

a) if more than 80% of friends of $x$ use Redmi 2A, and
b) none of the friends of $x$ gave Redmi 2A a bad rating, then the chances are that $x$ may like Redmi 2A
Extending graph patterns with first-order logic

- \( Q(x, y) \): edges labelled with counting quantifiers
  - existential quantifiers
  - universal quantifiers
  - negation

Balancing expressive power and complexity

- Input: A pattern \( Q \), and a graph \( G \)
- Question: Does there exist a match of \( Q \) in \( G \)?

The quantified matching problem is

- DP-complete in general
- NP-complete in the absence of negation

Practical to discover and apply GPARs

W. Fan, Y. Wu, and J. Xu. *Adding counting quantifiers to graph patterns*. SIGMOD 2016

Parallel scalable algorithms for quantified matching
Graph Pattern Association Rules (GPAR)

- \( R(x, y): \ Q(x, y) \implies p(x, y) \)
  - \( Q \): a graph pattern
  - \( x, y \): two variables for entities
  - \( p \): a predicate

- **Semantics:** graph pattern matching via subgraph isomorphism
  - support, confidence

- **GPAR discovery:** top-\( k \) diversified GPARs pertaining to \( p(x, y) \)
  with support above \( \sigma \), from a social graph \( G \)

- **Identifying potential customers:** the set of entities identified by
  GPARs in social graph \( G \) with confidence above \( \eta \)

---

*Parallel scalable algorithms in large social networks*

**W. Fan, X. Wang, Y. Wu, and J. Xu.**

*Association rules with graph patterns*,

*VLDB 2015*
Functional dependencies for graphs
Functional dependencies

Found in every database textbook: $R(X \rightarrow Y)$
✓ $X$ and $Y$: attributes of relation $R$
✓ $X$ attributes uniquely determine $Y$ attributes

Primitive constraints for relations
✓ Specify a fundamental part of the semantics of the data
✓ Conditional functional dependencies (CFDs): instrumental to capturing inconsistencies – data quality

The need is more evident for graphs:
✓ Specify the semantics in the absence of schema for graphs
✓ Detect inconsistencies in knowledge bases
✓ Catch spams in social networks

However, functional dependencies are not well studied for graphs
Inconsistencies in knowledge base

- DBPedia: John Brown is put as both a child and a parent of Owen Brown
- YAGO3: Saint Petersburg is labelled as a city in both Russia and Florida
- MKNF: It is marked that all birds can fly, and that penguins are birds

**New challenges introduced by graphs**

- A functional dependency $X \rightarrow Y$, to specify attribute dependency
- A topological constraint, to specify “the scope” of $X \rightarrow Y$

**A departure from our familiar functional dependencies**
functional dependency for graphs

✓ **GFD: Q(X → Y)**
  - Q: a graph pattern, to specify topological constraint
  - X, Y: sets of literals x.A = c, x.A = y.B, for vertices x and y, attributes A and B of the vertices, and constant c

✓ John Brown is put as both a child and a parent of Owen Brown

✓ Q(∅ → false), i.e., Q(x.A = c → x.A = d) for distinct c and d

A nontrivial extension of traditional functional dependencies
catching inconsistencies

✓ Saint Petersburg is labelled as a city in both Russia and Florida

✓ Q( ∅ → y.name = z.name)

Must be the same place

A combination of pattern and attribute dependencies
Generic property

- all birds can fly, and penguins are birds

Q( ∅ → x.A = y.A)
  - y inherits all the properties of x if y is_a x
  - wildcard _: any entity

A node in Q must have attribute A
Q( ∅ → x.A = x.A)

Specify the fundamental semantics of the data
Conditional functional dependencies (CFDs)

✓ R(country = UK, zip → street)

✓ Q( X → Y)
  - X: x.country = UK, y.country = UK, x.zip = y.zip
  - Y: x.street = y.street

✓ Traditional functional dependencies and conditional functional dependencies are special cases of GFDs

Catch inconsistencies in graphs
Catching spams

a) if $x'$ is a confirmed fake account,
b) both $x$ and $x'$ like $k$ blogs, and
c) if $x'$ posts blog $y_1$, $x$ posts $y_2$, $y_1$ and $y_2$ have the same keyword $c$,
then $x$ is also fake

$\checkmark$ $Q( X \rightarrow Y): X: x'.fake = true, z.keyword = c, z'.keyword = x$
$Y: x.fake = true$

Beyond inconsistency detection
Semantics

✓ A graph G satisfies $Q(X \rightarrow Y)$ if for all isomorphic mapping $h$ from a subgraph G to Q
  • if $h(X)$ is satisfied, then $h(Y)$ is satisfied

✓ Semi-structured nature: a vertex $x$ does not necessarily have a attribute $A$
  • if $x.A$ is in $X$ but $x$ does not have $A$, $h(X)$ is trivially true
  • if $x.A$ is in $Y$ but $x$ does not have $A$, $h(Y)$ is trivially false

✓ We can enforce $x$ to have an attribute $A$

✓ But we cannot enforce finite domains for an attributes, eg, Boolean, since G does not come with a schema

Coping with the semi-structured nature of graphs

Limited type constraint
Classical problems for GFDs

✓ Satisfiability
  • Input: A set $\Sigma$ of GFDs
  • Question: does there exists G that satisfies $\Sigma$?

✓ Implication
  • Input: A set $\Sigma$ of GFDs, and another GFD $\phi$
  • Question: does $\Sigma$ imply $\phi$? That is, for all graphs G, if G satisfies $\Sigma$, then G satisfies $\phi$

✓ Complexity bounds
  • The satisfiability problem is coNP-complete
  • The implication problem is NP-complete

No more expensive than their relational counterparts
Inconsistency detection in graphs

Input: A set $\Sigma$ of GFDs, and a graph $G$

Question: find all violations of $\Sigma$ in $G$, i.e., subgraphs $G'$ in $G$ such that there exists a CFD $Q(X \rightarrow Y)$ in $\Sigma$, where

- $G'$ is isomorphic to $Q$, and
- $G'$ does not satisfy $X \rightarrow Y$

Complexity: NP-complete

Positive: there exist parallel scalable algorithms for error detection

It is feasible to detect errors in large graphs with GFDs!

W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs, SIGMOD 2016

A first step towards a constraint theory for graphs
Summing up
Querying big data

Theory: conventional query paradigm no longer suffices
- a new paradigm: querying big data with constrained resources
- fundamental issues: model and complexity bounds

Systems: provide small companies with big data services
- BEAS: querying big relations with constrained resources

Application: social media marketing
- Association rules with graph patterns
- Identifying potential customers with counting quantifiers

Functional dependencies for graphs: theory and practice

One step further towards a practical solution to big data
References

Theory for querying big data


Techniques for querying big data:

References


Applications

✓ W. Fan, Y. Wu, and J. Xu. Adding counting quantifiers to graph patterns. SIGMOD 2016

✓ W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. SIGMOD 2016
