PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Perspective

Cite this article: Fan W. 2019 Making big data small. *Proc. R. Soc. A* **475**: 20190034.
http://dx.doi.org/10.1098/rspa.2019.0034

Received: 17 January 2019
Accepted: 29 March 2019

Subject Areas:
software

Keywords:
big data, database systems, structured query language, bounded evaluation, approximate query answering

Author for correspondence:
Wenfei Fan
e-mail: wenfei@inf.ed.ac.uk

Making big data small

Wenfei Fan[1,2,3]

[1]University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK
[2]Beihang University, 37 Xue Yuan Road, Haidian District, Beijing 100191, People's Republic of China
[3]Shenzhen Institute of Computing Sciences, Shenzhen University, Room 1001, Building 26, Hongshan 6979, Minbao Road, Longhua District, Shenzhen 518000, People's Republic of China

WF, 0000-0001-5149-2656

Big data analytics is often prohibitively costly and is typically conducted by parallel processing with a cluster of machines. Is big data analytics beyond the reach of small companies that can only afford limited resources? This paper tackles this question by presenting Boundedly EvAlable SQL (BEAS), a system for querying big relations with constrained resources. The idea is to make big data small. To answer a query posed on a dataset, it often suffices to access a small fraction of the data no matter how big the dataset is. In the light of this, BEAS answers queries on big data by identifying and fetching a small set of the data needed. Under available resources, it computes exact answers whenever possible and otherwise approximate answers with accuracy guarantees. Underlying BEAS are principled approaches of bounded evaluation and data-driven approximation, the focus of this paper.

1. Introduction

Big data analytics appeals for a departure from the classical computational complexity theory and conventional query evaluation paradigm. Textbooks tell us that a computational problem is *tractable* if its cost can be expressed as a polynomial in the size n of the input [1]. When it comes to big data, however, it may no longer be the case. As an example, let us consider queries on relational datasets, i.e. tables of records (*a.k.a.* tuples). It was reported that it took days to join two tables of 100 million tuples each [2]. While join is a routine operation in relational queries and its cost is in $O(n^2)$ time on tables of n tuples, it can no longer be considered 'tractable' if it takes days on big relations.

THE ROYAL SOCIETY
PUBLISHING

That is, computational problems that are labelled tractable in the classical complexity theory may become infeasible or 'intractable' in the context of big data [3].

The industrial solution to big data analytics is typically by parallel computation with a cluster of machines. When data grows big, it scales out the cluster by adding more machines. The solution assumes the *parallel scalability*, i.e. the computation would get faster when given more machines. However, few parallel algorithms in the literature offer this performance guarantee. Worse still, some computational problems are not parallel scalable, i.e. there exist no algorithms for these problems that guarantee to substantially reduce running time by adding more machines; for instance, no parallel algorithm for graph simulation, an $O(n^2)$ problem for social network analyses, is able to further reduce running time by adding more processors, after the number of processors used reaches a point [4] (see [5] for more real-life examples). Furthermore, even when dealing with computational problems that are parallel scalable, small businesses typically have limited resources and cannot afford large-scale parallel computation.

Is big data analytics beyond the reach of small companies with limited resources?

We tackle this question by presenting an alternative solution, as an effort to provide small companies with a capacity of big data analytics under constrained resources. We focus on relational query answering, since relational data accounts for 75% of business data in the real world [6], relational databases dominate big data [7], and analytics of the data is typically carried out with queries expressed in SQL (structured query language, the standard query language for relational databases). Given an SQL query Q and a relational dataset D, our job is to compute the answer to Q in D, denoted by $Q(D)$, which is a set of tuples. The query answering problem is non-trivial. It is NP-complete (intractable) to decide whether a given tuple is in the answer $Q(D)$ when Q is in SPC, a special case of SQL defined with *selection*, *projection* and *Cartesian product* (*join*) operators only.

The idea is to make big data small, based on the following observation. Given a query Q posed on a dataset D, in practice it often suffices to compute $Q(D)$ by accessing a small fraction D_Q of D, even when D is big. In the light of this, we propose (a) *bounded evaluation* [8–12]. Given a query Q, it identifies D_Q such that $Q(D_Q) = Q(D)$ and computes the exact answer to Q in D by accessing small data D_Q; moreover, it fetches D_Q of a bounded size when possible. When exact answers are beyond reach of a bounded fraction of D, we propose (b) a *data-driven approximation* scheme [13]. Also identifying and fetching a small fraction D_Q of D, it computes $Q(D_Q)$ as approximate answer to Q in D and offers a deterministic accuracy bound. Both (a) and (b) are carried out under available limited resources and reduce queries on big data to computations on small data.

As proof of concept, we have developed a prototype system BEAS (Boundedly EvAlable SQL) [14], which implements the new query evaluation paradigm to answer SQL queries under constrained resources. Our industry collaborators have evaluated BEAS with call-detailed-record (CDR) queries and found that BEAS is able to compute exact answers to more than 90% of their CDR queries, reduce big datasets from PB (10^{15}) to GB (10^9) and improve the performance by orders of magnitude [15].

Organization. In the remainder of the paper, we give a brief introduction to bounded evaluation (§2) and the data-driven approximation scheme (§3). Putting these together, we present BEAS, our resource-bounded query evaluation system (§4). We focus on SQL queries on big data in this paper, which account for a large part of big data analytics.

2. Bounded evaluation

Given an SQL query Q posed on a big dataset D, bounded evaluation [9] aims to compute the exact answer $Q(D)$ to Q in D by accessing only a bounded subset D_Q of D that includes necessary information for answering Q in D, instead of the entire D. The idea is simple, but it is non-trivial to identify D_Q. To this end, it makes use of an access schema $\mathcal{A}$, which is a set of access constraints, i.e. a combination of simple cardinality constraints and their associated indices on the data.

Bounded evaluation. Under an access schema $\mathcal{A}$, query Q is *boundedly evaluable* if for each dataset D that conforms to $\mathcal{A}$, there exists a fraction $D_Q \subseteq D$ such that

— $Q(D_Q) = Q(D)$, i.e. it suffices to compute the *exact answer* $Q(D)$ by only using Q_D; and
— the time for identifying D_Q and the size $|D_Q|$ of D_Q are determined by the query Q and the schema $\mathcal{A}$ only. That is, the cost of computing $Q(D_Q)$ is independent of the size $|D|$ of D.

Intuitively, query Q is boundedly evaluable under $\mathcal{A}$ if $Q(D)$ can be computed by accessing a bounded fraction D_Q of D, no matter how big the dataset D grows. We identify D_Q by reasoning about the cardinality constraints in $\mathcal{A}$ and fetch D_Q by using the indices in $\mathcal{A}$. A dataset D conforms to $\mathcal{A}$ if D satisfies the cardinality constraints of $\mathcal{A}$ and has indices of $\mathcal{A}$ built on it.

Example 2.1. Consider a database schema $\mathcal{R}_0$ that consists of four relations (tables):

(a) person(pid, city, country), each tuple specifies a person pid who lives in city of country;
(b) friend(pid, fid), stating that person fid is a friend of person pid;
(c) restaurant(rid, name, rating, price, street, city, country), specifying the name, rating, average cost per person and the address of a restaurant and
(d) visit(pid, rid, dd, mm, yy), saying that person pid visited restaurant rid on date (dd, mm, yy).

Consider a query Q_1 to *find all restaurants in London that were rated A and were visited in 2018 by one of my friends who lives in London*. This is a personalized search query taken from Graph Search of Facebook [16]. Written in SQL, query Q_1 can be expressed as follows:

select restaurant.rid, restaurant.name
from friend, person, visit, restaurant
where friend.pid = p_0 **and** friend.fid = person.pid and person.city = 'London' **and**
friend.fid = visit.pid **and** visit.yy = 2018 **and**
visit.rid = restaurant.rid **and** restaurant.rating = 'A' **and** restaurant.city = 'London'

where p_0 indicates 'me' for personalized search. The query uses three *join* operations: (a) a join between tables friend and person to find friends of p_0; (b) one between table visit and the set (table) of friends of p_0 retrieved in (a) above, to find rids visited 'by my friends' and (c) another join between table restaurant and the set of the rids selected, to find attributes name, city and rating of these restaurants. It employs conditions to *select* friends of p_0 who lives in London and restaurants in London that were rated A and were visited in 2018. Finally, it returns a table of tuples with rid and name attributes of restaurant, by applying *projection* on the intermediate result.

An instance D_0 of database schema $\mathcal{R}_0$ may be 'big', e.g. Facebook has billions of users and trillions of friend links [17]. On such a dataset D_0, it is prohibitively costly to compute $Q_1(D_0)$.

However, we can do better by making use of access constraints. An example access schema $\mathcal{A}_1$ consists of the following five access constraints:

— ψ_1: friend(pid $\rightarrow$ fid, 5000);
— ψ_2: visit(yy $\rightarrow$ (dd, mm, yy), 366);
— ψ_3: visit((pid, dd, mm, yy) $\rightarrow$ rid, 1);
— ψ_4: person(pid $\rightarrow$ city, 1) and
— ψ_5: restaurant(rid $\rightarrow$ (name, city, rating), 1).

Here, ψ_1 is a cardinality constraint imposed by Facebook [17], which puts a limit of 5000 friends per person; ψ_2 says that each year has at most 366 days; ψ_3 states that on a given day, each person dines out at most once; and ψ_4 is a simple constraint known as a key, stating that the pid of a person uniquely determines the city where the person lives, similarly for ψ_5. An index is built for ψ_1 such that given a pid, it returns all friend fids of the pid from friend, i.e. ψ_1 is a combination of the

cardinality constraint (5000 fids for each pid) and the index, similarly for ψ_2–ψ_5. These are simple cardinality constraints commonly found in the real world, along with their associated indices.

Under the set $\mathcal{A}_1$ of access constraints, query Q_1 is boundedly evaluable. Indeed, we can (1) first identify and fetch at most 5000 friend fids of person p_0 from relation friend by using the index for ψ_1 and (2) for each fid fetched, we do the following: (a) get 1 city where the person fid lives from relation person by using the index for ψ_4; (b) using the index for ψ_2 and constant 2018, fetch at most 366 (dd, mm, yy) partial tuples from table visit; (c) putting the fid together with each (dd, mm, yy), fetch at most 366 rids from visit by using the index for ψ_3 and (d) for each of the rids found, retrieve at most 1 (name, city, rating) tuple from the restaurant table by using the index for ψ_5. In total, we fetch a set D_Q consisting of $5000 \times 2 + 366 + 5000 \times 366 \times 2$ tuples, instead of trillions. The set D_Q contains all the data that are needed for computing $Q_1(D_0)$, and hence, it suffices to answer query Q_1 in the big dataset D_0 by using the set D_Q of retrieved data only. Better yet, under the access schema $\mathcal{A}_1$, the size of D_Q remains stable no matter how big dataset D_0 grows.

Bounded evaluation suggests that given an application that requires querying big datasets, we can first discover an access schema $\mathcal{A}$ offline, based on an analysis of historical queries. We then take SQL queries Q online and check whether Q is boundedly evaluable under $\mathcal{A}$. If so, query Q is guaranteed to be answered on big datasets by accessing a bounded amount of data.

Effective algorithms are in place for discovering access schema from historical queries, incrementally maintaining the access schema in response to updates, checking the bounded evaluability of input queries and generating query plans for bounded queries by accessing a bounded amount of data. As an example, below we consider how to check the bounded evaluability.

Deciding bounded evaluability. A challenge is that it is undecidable to check whether an SQL query is boundedly evaluable under an access schema $\mathcal{A}$ [9]. The undecidability remains intact for relational algebra (RA), denoted by RA, which is SPC extended with set *union* and *difference*, and is the first-order logic fragment of SQL. To cope with this, an *effective syntax* $\mathcal{L}$ was developed [12]. That is, $\mathcal{L}$ is a class of RA queries such that under access schema $\mathcal{A}$,

(a) an RA query Q is boundedly evaluable if and only if it is equivalent to a query Q' in $\mathcal{L}$ and
(b) it takes PTIME (polynomial time) in the size $|Q|$ of query Q and the size $|\mathcal{A}|$ of access schema $\mathcal{A}$ to check whether Q is in $\mathcal{L}$, reducing the problem to syntactic checking.

That is, $\mathcal{L}$ identifies the *core* subclass of boundedly evaluable RA queries under access schema $\mathcal{A}$, without sacrificing their expressive power up to equivalence. This is analogous to how database management systems (DBMSs) support range-safe relational calculus queries, which is undecidable [18]. Based on the effective syntax, we can efficiently check whether a query Q is boundedly evaluable, and if so, generate a query plan to answer Q by accessing a bounded amount of data [12].

The unique features of bounded evaluation. The foundation of bounded evaluation was established in [8–10]. The theory was first put in action for SPC [11] and then extended to RA [12]. A prototype system was developed in [14] and offers the following:

(1) *Bounded evaluability*. It is able to check whether an input SQL query Q is boundedly evaluable, and if so, answer Q by accessing a bounded amount of data from a (possibly big) dataset.

(2) *Plug and play*. Bounded evaluation can be readily built on top of commercial DBMSs and provides the DBMSs with a convenient capability of querying big relations under limited resources.

(3) *Join reduction*. Access constraints provide us with new optimization rules, to replace costly join operations with simple data fetch by leveraging their associated indices at the query level.

We found that about 77% of SPC queries [11] and 67% of SQL queries [12] are boundedly evaluable. Our industry collaborators found that more than 90% of their CDR queries are boundedly evaluable.

The study of bounded evaluation is among the first efforts to compute exact answers to queries in big datasets under limited resources. In contrast to commercial DBMSs, it decides whether an

input query is boundedly evaluable, and if so, it generates a bounded query plan, by reasoning about access schema. Access schema is implemented with indices, but as opposed to conventional indices, it allows us to reason about cardinality constraints at the query level, reduce costly join operations and deduce bounded query plans. Moreover, conventional indices retrieve entire tuples [19]. In contrast, with access constraints, we fetch distinct partial tuples consisting of only attributes needed for answering a query. For instance, as shown in example (2.1), we only fetch the name, rating and city attributes of restaurant, but not price, street and country, since answering Q_1 does not need the latter. That is, bounded evaluation does not retrieve duplicated and unnecessary attributes or tuples; the redundancies get inflated rapidly with joins in conventional query evaluation.

3. Data-driven approximation

For queries Q that are boundedly evaluable under an access schema, we can answer Q by accessing a bounded amount of data. What can we do if query Q is not boundedly evaluable? Is it still possible to compute $Q(D)$ in a big dataset D under constrained resources? We tackle this question next.

Approximation. We propose a data-driven scheme for approximate query answering. It is parameterized with a *resource ratio* $\alpha \in (0, 1]$, indicating that our available resources can only afford to access an α-fraction of a big dataset D. Given ratio α, dataset D and an SQL query Q over D, the approximation scheme identifies $D_Q \subseteq D$ and computes $Q(D_Q)$ and a ratio $\eta \in (0, 1]$ such that

(1) $|D_Q| \leq \alpha |D|$, where $|D_Q|$ is measured in its number of tuples and
(2) $\mathsf{accuracy}(Q(D_Q), Q, D) \geq \eta$, where η is the deduced accuracy bound.

Intuitively, the scheme computes a set $Q(D_Q)$ of tuples as the approximate answer to query Q in big dataset D, by accessing at most $\alpha|D|$ tuples in the entire process. Thus, it can scale with D when D grows big by setting the ratio α small. Moreover, $Q(D_Q)$ assures a *deterministic* accuracy bound η [13]:

(a) for *each* tuple s in the approximate answer $Q(D_Q)$, there exists a tuple t in the exact answer $Q(D)$ that is within distance η of s, and conversely,
(b) for *each* tuple t in the exact answer $Q(D)$, there exists a tuple s in the approximate answer $Q(D_Q)$ that is within distance η of t.

That is, the set $Q(D_Q)$ consists only of 'relevant' answers, i.e. each tuple in $Q(D_Q)$ is a sensible answer to users, up to distance η; here, the distance between s and t is the sum of the pairwise distances of their attributes, which is in turn measured in terms of a distance function in the domain of the corresponding attribute (see an example shortly and [13] for a definition). Moreover, $Q(D_Q)$ 'covers' the exact answer, i.e. each tuple in the exact answer $Q(D)$ finds a tuple $s \in Q(D_Q)$ within distance η, and no tuple in $Q(D)$ is missed by $Q(D_Q)$ up to bound η. It finds sensible answers in users' interest and suffices for exploratory queries, e.g. real-time problem diagnosis on logs [20]. The larger the ratio α is, i.e. the more resources we can afford, the higher the accuracy bound η we can get.

As observed in a recent survey [21], approximate query answering is challenging, and little practical use has emerged from it. Nonetheless, we can make data-driven approximation work in practice by employing a mild extension of the access schema that we have seen earlier.

Example 3.1. Continuing with example 2.1, consider query Q_2 to *find me restaurants that cost at most £35 per person on average and are in a city where one of my friends lives*:

```
select restaurant.name, restaurant.price
from   restaurant, friend, person
```

where friend.pid = p_0 **and** friend.fid = person.pid **and**
person.city = restaurant.city **and** restaurant.price ≤ 35

Here, again p_0 indicates 'me'. The query is defined with two join operations, one between tables friend and person to find the cities where 'my friends' live, and the other between table restaurant and the set of 'my friends' to make sure that they are in the same city. It is costly to compute the exact answer $Q_2(D_0)$ in a big dataset D_0 with billions of person tuples and trillions of friend links.

Suppose that our available resources can afford to access at most $10^{-4} * |D_0|$ tuples, i.e. $\alpha = 10^{-4}$. That is, we have to reduce a dataset of PB size to a dataset of 100 GB, while guaranteeing a deterministic accuracy bound. Under these constraints, we can compute approximate answers to query Q_2 in D_0 based on data-driven approximation, by making use of an access schema $\mathcal{A}_2$, which includes ψ_1–ψ_5 of example (2.1), and in addition, the following extended access constraints:

— ϕ_1: restaurant(city $\rightarrow$ (name, price), 1, (e_n^1, e_p^1)),
...
— ϕ_m: restaurant(city $\rightarrow$ (name, price), 2^m, (e_n^m, e_p^m)), where $m = \lceil \log_2 M \rceil$.

Here, M is the maximum number of distinct restaurant tuples in dataset D_0 grouped by attribute city. Observe that there exists no constant bound on the number of restaurants in a city, and hence, it is hard to define access constraints along the same lines as ψ_1–ψ_5 of example (2.1). Instead, we build an index for each ϕ_i in $\mathcal{A}_2$ such that for each $i \in [1, m]$, given any city-value c, we can retrieve a set T of at most 2^i (name, price) values from D_0 by using the index. Moreover, for each restaurant tuple $t = (id, n, r, p, s, c, c_1)$ in D_0 (recall that such tuples are specified by (rid, name, rating, price, street, city, country)), there exists a tuple $(n', p') \in T$ such that the (name, price)-value (n, p) of t differs from (n', p') by distances at most (e_n^i, e_p^i); for instance, when $e_p^i = 4$, price p' returned is at most £4 more expensive than the real price p. That is, T represents (name, price) values that correspond to city-attribute c with at most 2^i tuples, subject to distances (e_n^i, e_p^i). Intuitively, the indices give us a hierarchical representation of relation restaurant with different resolutions $i \in [1, m]$. The higher the resolution level i is, the smaller the distances (e_n^i, e_p^i) are, and the more accurately the index for ϕ_i represents the data in D_0.

Under access schema $\mathcal{A}_2$, we can find restaurants of interest by accessing at most $\alpha|D_0|$ tuples as follows: (a) fetch a set T_1 of friend.fid's of p_0 by accessing at most 5000 friend tuples, using the access constraint ψ_1 of example (2.1); (b) for each fid in T_1, fetch 1 associated city from relation person by using the index for ψ_4, which yields a set T_2 of at most 5000 city values; (c) for each city c in T_2, fetch at most 2^{k_α} (name, price) pairs corresponding to c from relation restaurant by using the index for ϕ_{k_α}, where $k_\alpha = \lfloor \log_2(\alpha|D_0| - 10\,000) \rfloor$ and (d) return a set S of those (name, price) values with price at most $(35 + e_p^{k_\alpha})$, as approximate answer to Q_2 in D_0. The process accesses at most 5000 + 5000 + $2^{k_\alpha} \leq \alpha|D_0|$ tuples in total, instead of trillions.

The approximate answer S has a deterministic accuracy bound: (1) for each tuple (n, p) in the exact answer $Q(D_0)$, there exists a tuple (n', p') in S such that n' and p' are within distance $e_n^{k_\alpha}$ and $e_p^{k_\alpha}$ of n and p, respectively, and (2) for each tuple in S, its price p' exceeds 35 by at most $e_p^{k_\alpha}$, e.g. $e_p^{k_\alpha} = 4$ and $p' = 39$, and n' is the name of a restaurant. Moreover, the larger the resource ratio α is, the smaller the distances $e_p^{k_\alpha}$ and $e_n^{k_\alpha}$ are, and the more accurate the approximate answer S is.

It has been shown [13] that for any dataset D, there exists an access schema $\mathcal{A}$ such that for any resource ratio $\alpha \in (0, 1]$ and any SQL query Q over D, one can effectively identify a subset D_Q of D by reasoning about the constraints in $\mathcal{A}$ and deduce a deterministic accuracy bound η such that $|D_Q| \leq \alpha|D|$ and accuracy$(Q(D_Q), Q, D) \geq \eta$. Moreover, the larger the resource ratio α is, the higher the accuracy bound η is. The access schema $\mathcal{A}$ can be effectively computed offline and efficiently maintained online in response to updates to dataset D [13].

Unique features of data-driven approximation. There has been a host of work on approximate query answering, typically based on one of the following two approaches (see [21,22] for surveys): (a) synopsis [23–27] or (b) dynamic sampling, e.g. [20,28–30]. The first approach computes

an one-size-fit-all synopsis D' of a dataset D and uses D' to answer all queries posed on D. A representative method of the second approach is BlinkDB [20]. Assuming predictable QCSs (query column sets), i.e. 'the frequency of columns used for grouping and filtering does not change over time', BlinkDB selects samples from historical QCS patterns and caches them as views. It targets aggregate queries, i.e. queries to compute max, min, count, sum and average values grouped by certain attributes. It answers such aggregate queries by using the samples instead of accessing the original datasets and offers probabilistic error rates for the approximate answers computed.

As opposed to the prior approaches, the data-driven approximation scheme makes use of an access schema to identify an α-fraction D_Q of dataset D. It offers the following:

(1) *Unpredictable queries*. The scheme does not assume the availability of any prior knowledge about user queries Q. By contrast, previous approaches make various assumptions on future queries, e.g. they assume that workloads, query predicates or QCSs of future queries are known in advance.

(2) *Generic queries*. The scheme is able to handle generic SQL queries, aggregate or not. For instance, query Q_2 in example (3.1) is not an aggregate query. By contrast, previous approaches 'substantially limit the types of queries they can execute' [20] and typically focus on aggregate queries.

(3) *Deterministic accuracy bound*. As remarked earlier, the data-driven scheme proposes an accuracy measure in terms of both relevance and coverage, to ensure that each tuple in the approximate answer computed is sensible and all tuples in the exact answer are covered. By contrast, previous approaches provide either no accuracy guarantee at all or probabilistic error rates for aggregate queries only. Such error rates do not tell us how 'good' each approximate answer is, which is one of the reasons why approximate query answering has not been widely used in practice.

Our preliminary study using real-life data finds that the data-driven approximation scheme computes approximate answers to SQL queries, aggregate or not, with accuracy $\eta \geq 0.82$, even when α is as small as 5.5×10^{-4} [13]. That is, it reduces D of PB size to D_Q of 550 GB.

4. A resource-bounded query evaluation framework

Putting bounded evaluation and data-driven approximation together, we develop BEAS (Boundedly EvAluable Sql), a system for querying big relations under constrained resources.

BEAS. For a big dataset D in an application, BEAS takes a *resource ratio* $\alpha \in (0, 1]$ as a parameter, which is estimated by an analysis of available resources and the size of dataset D. It discovers an access schema $\mathcal{A}$ offline, by striking a balance between the speedup of query evaluation and the space cost introduced by the indices in the access schema $\mathcal{A}$. Then, BEAS answers SQL queries posed on dataset D online. Given an SQL query Q, BEAS works as follows:

(1) it first checks whether query Q is boundedly evaluable under $\mathcal{A}$, i.e. the exact answer $Q(D)$ can be computed by accessing $D_Q \subseteq D$ such that $|D_Q|$ is independent of the size $|D|$ of D;
(2) if so, it computes the exact answer $Q(D)$ by accessing a bounded fraction D_Q of D;
(3) otherwise, BEAS identifies a fraction D_Q of D with $|D_Q| \leq \alpha|D|$ and computes $Q(D_Q)$ with a deterministic accuracy bound η, based on data-driven approximation.

That is, under the resource constraint α, BEAS computes exact answers $Q(D)$ when possible, and approximate answers $Q(D_Q)$ otherwise with accuracy η. In the entire process, it accesses a small fraction of data that can be afforded by available resources. BEAS also supports incremental maintenance of the indices in $\mathcal{A}$ in response to updates to dataset D and guarantees that the maintenance cost is determined by the size of changes, not by the size of dataset D [11,13].

Unique features of BEAS. BEAS departs from commercial DBMS by promoting an unconventional query evaluation paradigm. It is unique in the following:

(1) *Querying big relations*. It is parameterized with a *resource ratio* α, i.e. it is able to scale with arbitrarily large datasets D by adjusting ratio α based on available resources.

(2) *Separating exact answers from approximate answers*. By effectively deciding whether an input query is *boundedly evaluable* under an access schema, BEAS is capable of telling its users whether the answers to their queries are exact or approximate. Moreover, for approximate answers, it provides deterministic accuracy bounds η in terms of both relevance and coverage.

(3) *Generic*. BEAS is able to answer SQL queries that are *unpredictable*, without assuming prior knowledge about future queries, and *generic*, no matter whether the queries are aggregate or not.

(4) *Ease of implementation*. BEAS can be built on top of commercial DBMS and provide the DBMS with an immediate capacity to query big relations under constrained resources. It offers its own optimization strategies and is also able to further improve its query plans, bounded or approximate, by making use of existing DBMS optimizers that have been developed for decades.

In light of these, BEAS is promising for providing small companies with the capability of big data analytics under limited resources. It can also help big companies to reduce the cost and improve efficiency [15]. In particular, for computational problems that are not parallel scalable, bounded evaluation and data-driven approximation offer a feasible solution that does not rely on large-scale parallel computation. This said, bounded evaluation and data-driven approximation can be parallelized to benefit from parallel processing provided that resources are available.

The idea of resource-bounded query answering is not limited to querying big relations. It has been shown that bounded evaluation improves the performance of graph pattern matching via subgraph isomorphism, an intractable problem [1] that is widely used in social media marketing, knowledge base extension and graph data cleaning, by four orders of magnitude on average [31]. For personalized social search via subgraph isomorphism, data-driven approximation retains 100% accuracy (i.e. $\eta = 1$) when α is as small as 1.5×10^{-6} [32], i.e. when processing graphs G of 1 PB, they access only 15 GB of data, i.e. reducing G from PB to GB while retaining high accuracy.

Data accessibility. This article has no additional data.

Competing interests. I declare I have no competing interests.

Funding. W.F. is supported in part by ERC 652976, Royal Society Wolfson Research Merit Award WRM/R1/180014, NSFC 61421003, EPSRC EP/M025268/1, Foundation for Innovative Research Groups of NSFC, Joint Lab between Edinburgh and Huawei, Beijing Advanced Innovation Centre for Big Data and Brain Computing and Shenzhen Institute of Computing Sciences.

References

1. Papadimitriou CH. 1994 *Computational complexity*. Reading, MA: Addison-Wesley.
2. Stack Overflow. 2017 SQL: Inner joining two massive tables. See http://stackoverflow.com/questions/1750001/sql-inner-joining-two-massive-tables.
3. Fan W, Geerts F, Neven F. 2013 Making queries tractable on big data with preprocessing. *Proc. VLDB Endowment* **6**, 685–696. (doi:10.14778/2536360.2536368)
4. Fan W, Wang X, Wu Y. 2014 Distributed graph simulation: impossibility and possibility. *Proc. VLDB Endowment* **7**, 1083–1094. (doi:10.14778/2732977.2732983)
5. Xie C, Chen R, Guan H, Zang B, Chen H. 2015 SYNC or ASYNC: time to fuse for distributed graph-parallel computation. In *PPoPP*, pp. 194–204. See https://doi.org/10.1145/2688500.2688508.
6. Unisphere Research. 2016 See http://www.unisphereresearch.com/Content/ReportDetail.aspx?IssueID=6559.
7. Asay M. 2016 NoSQL keeps rising, but relational databases still dominate big data. See https://www.techrepublic.com/article/nosql-keeps-rising-but-relational-databases-still-dominate-big-data/.
8. Fan W, Geerts F, Libkin L. 2014 On scale independence for querying big data. In *PODS*. See https://doi.org/10.1145/2594538.2594551.
9. Fan W, Geerts F, Cao Y, Deng T. 2015 Querying big data by accessing small data. In *PODS*. See https://doi.org/10.1145/2745754.2745771.
10. Cao Y, Fan W, Geerts F, Lu P. 2016 Bounded query rewriting using views. In *PODS*. See https://doi.org/10.1145/2902251.2902294

11. Cao Y, Fan W, Wo T, Yu W. 2014 Bounded conjunctive queries. *Proc. VLDB Endowment* **7**, 1231–1242. (doi:10.14778/2732977)
12. Cao Y, Fan W. 2016 An effective syntax for bounded relational queries. In *SIGMOD*. See https://doi.org/10.1145/2882903.2882942.
13. Cao Y, Fan W. 2017 Data driven approximation with bounded resources. *Proc. VLDB Endowment* **10**, 973–984. (doi:10.14778/3099622)
14. Cao Y, Fan W, Wang Y, Yuan T, Li Y, Chen LY. 2017 BEAS: bounded evaluation of SQL queries. In *SIGMOD*, pp. 1667–1670. See https://doi.org/10.1145/3035918.3058748
15. University of Edinburgh. 2017 Huawei deal to advance expertise in data science. See http://www.ed.ac.uk/news/2017/huawei-deal-to-advance-expertise-in-data-science.
16. Facebook. 2013 Introducing graph search. See https://en-gb.facebook.com/about/graphsearch.
17. Grujic I, Bogdanovic-Dinic S, Stoimenov L. 2014 Collecting and analyzing data from e-government facebook pages. In *ICT Innovations*.
18. Abiteboul S, Hull R, Vianu V. 1995 *Foundations of databases*. Reading, MA: Addison-Wesley.
19. Ramakrishnan R, Gehrke J. 2000 *Database management systems*. New York, NY: McGraw-Hill Higher Education.
20. Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I. 2013 BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*. See https://doi.org/10.1145/2465351.2465355.
21. Chaudhuri S, Ding B, Kandula S. 2017 Approximate query processing: no silver bullet. In *SIGMOD*, pp. 511–519. See https://doi.org/10.1145/3035918.3056097.
22. Cormode G, Garofalakis MN, Haas PJ, Jermaine C. 2012 Synopses for massive data: samples, histograms, wavelets, sketches. *Found. Trends Databases* **4**, 1–294. (doi:10.1561/1900000004)
23. Acharya S, Gibbons PB, Poosala V. 2000 Congressional samples for approximate answering of group-by queries. In *SIGMOD*. See https://doi.org/10.1145/342009.335450.
24. Ioannidis YE, Poosala V. 1999 Histogram-based approximation of set-valued query-answers. In *VLDB*. See https://doi.org/10.1109/69.250091.
25. Jagadish HV, Koudas N, Muthukrishnan S, Poosala V, Sevcik KC, Suel T. 2009 Optimal histograms with quality guarantees. In *VLDB*. See https://doi.org/10.1145/130283.130335.
26. Chakrabarti K, Garofalakis MN, Rastogi R, Shim K. 2001 Approximate query processing using wavelets. *VLDB J.* **10**, 199–223. (doi:10.1007/s007780100049)
27. Cormode G, Garofalakis M. 2005 Sketching streams through the net: distributed approximate query tracking. In *VLDB*. See https://doi.org/10.1016/j.jalgor.2003.12.001.
28. Babcock B, Chaudhuri S, Das G. 2003 Dynamic sample selection for approximate query processing. In *SIGMOD*. See https://doi.org/10.1145/872757.872822.
29. Ding B, Huang S, Chaudhuri S, Chakrabarto K, Wang C. 2016 Sample + Seek: approximating aggregates with distribution precision guarantee. In *SIGMOD*. See https://doi.org/10.1145/2882903.2915249.
30. Kandula S, Shanbhag A, Vitorovic A, Olma M, Grandl R, Chaudhuri S, Ding B. 2016 Quickr: lazily approximating complex ad-hoc queries in big data clusters. In *SIGMOD*. See https://doi.org/10.1145/304181.304581.
31. Cao Y, Fan W, Huang R. 2015 Making pattern queries bounded in big graphs. In *ICDE*. See https://doi.org/10.1109/ICDE.2015.7113281.
32. Fan W, Wang X, Wu Y. 2014 Querying big graphs within bounded resources. In *SIGMOD*. See https://doi.org/10.1145/2588555.2610513.