Universalizing Approximate Query Processing
Universal

Approximate Query Processing
Universal
Approximate Query Processing
What is Approximate Query Processing (AQP)?

I/O + Computation → Exact Answer
What is Approximate Query Processing (AQP)?

I/O

Less I/O

Computation

Less Computation

Exact Answer

Approximate Answer
Why AQP?

Higher Productivity

Numerous studies:

A latency >2 seconds is no longer interactive and negatively affects creativity!
Why AQP?

**Higher Productivity**

Numerous studies:

*A latency >2 seconds* is no longer interactive and negatively affects creativity!

**Lower Cost (Time + Resources)**

Human time: Money

Machine time: No one loves their EC2 bill!
Why AQP?

Higher Productivity

Numerous studies:

* A latency > 2 seconds is no longer interactive and negatively affects creativity!

Lower Cost (Time + Resources)

* Human time: Money
* Machine time: No one loves their EC2 bill!

Jeff Bezos
AQP research in academia

- 1984: Approximate count estimator
- 1985: Double sampling
- 1991: Online aggregation
- 1994: Selectivity estimation on random samples
- 1996: AQUA, Ripple join
- 1997: Dynamic sample selection
- 2000: STRAT
- 2003: Bootstrap for AQP
- 2005: SMS join
- 2006: MapReduce Online, COSMOS
- 2007: Optimized stratified, Scalable with DBO
- 2010: SciBORQ
- 2011: QuickR, Seek+Sample, Wander join
- 2013: BlinkDB
- 2016: DBLearning
AQP research in academia

1984
Approximate count estimator
1985
Double sampling

1991
Online aggregation
1996
Selectivity estimation on random samples
1997
AQUA, Ripple join
1999
Dynamic sample selection
2000
STRAT
2001
Bootstrap for AQP
2003
Ripple join
2005
SMS join
2006
MapReduce Online, COSMOS
2009
Optimized stratified, Scalable with DBO
2011
SciBORQ
2013
BlinkDB
2016
QuickR, Seek+Sample, Wander join
2017
DBLearning
2020

35 years of research, little industry adoption
AQP is hard to adopt

AQP typically requires **significant** modifications of DBMS internals

- **Error estimation:** [BlinkDB ‘13], [G-OLA ’15], ...
- **Query evaluation:** [Online ‘97], [Join Synopses ‘99], ...
- **Relational operators:** [ABM ‘14], ...
AQP is hard to adopt

AQP typically requires **significant** modifications of DBMS internals

- **Error estimation:** [BlinkDB ‘13], [G-OLA ’15], ...
- **Query evaluation:** [Online ‘97], [Join Synopses ‘99], ...
- **Relational operators:** [ABM ‘14], ...

**Traditional DBMS vendors**

- Stable codebase, reluctant to make major changes
- Slow in adopting ANYTHING :-)
AQP is hard to adopt

AQP typically requires **significant** modifications of DBMS internals

- **Error estimation:** [BlinkDB ‘13], [G-OLA ’15], ...
- **Query evaluation:** [Online ‘97], [Join Synopses ‘99], ...
- **Relational operators:** [ABM ‘14], ...

**Traditional DBMS vendors**

- Stable codebase, reluctant to make major changes
- Slow in adopting ANYTHING :-(

**Newer SQL-on-Hadoop systems:** implementing standard features
AQP is hard to adopt

AQP typically requires significant modifications of DBMS internals

- Error estimation: [BlinkDB ‘13], [G-OLA ’15], ...
- Query evaluation: [Online ‘97], [Join Synopses ‘99], ...
- Relational operators: [ABM ‘14], ...

Traditional DBMS vendors

- Stable codebase, reluctant to make major changes
- Slow in adopting ANYTHING :-)

Newer SQL-on-Hadoop systems: implementing standard features

Users won’t abandon their existing DBMS just to use AQP.
Built-in AQP functions in OLAP engines

- **Amazon Redshift**: `APPROXIMATE PERCENTILE_DISC`
- **Oracle**: `approx_count_distinct`, `approx_percentile`
- **Apache Spark**: `approxCountDistinct`, `approxQuantile`
- **Cloudera Impala**: `NDV`, `APX_MEDIAN`
Built-in AQP functions in OLAP engines

- Amazon Redshift: APPROXIMATE PERCENTILE_DISC
- Oracle: `approx_count_distinct`, `approx_percentile`
- Apache Spark: `approxCountDistinct`, `approxQuantile`
- Cloudera Impala: NDV, APX_MEDIAN

Good progress!
But, too little, too slow
Built-in AQP functions in OLAP engines

**Limitations**

1. Good *only when* the data does not fit in memory
2. Good *only for* flat queries: *no error propagation*
3. Applicable *only for* order statistics: *no support for UDAs or arithmetic aggregates*

**Good progress!**

*But, too little, too slow*
Built-in AQP functions in OLAP engines

Limitations

1. Good only when the data does not fit in memory
2. Good only for flat queries: no error propagation
3. Applicable only for order statistics: no support for UDAs or arithmetic aggregates

Need for complete AQP solutions that are easy to adopt
Our proposal: Universal AQP

user/app

Exact Result

SQL

SQL DB
Our proposal: Universal AQP

Thin AQP layer

user/app

AQP

SQL DB
Our proposal: Universal AQP

SQL

user/app

select avg(price) from sales where channel = 'online'

Thin AQP layer

SQL DB
Our proposal: Universal AQP

```
select avg(price) from sales where channel = 'online'
```

```
select avg(a1), std(a1)
from (select avg(price) as a1 from sales where channel = 'online' group by sid) t1
```
Our proposal: Universal AQP

Thin AQP layer

Approx Result + Error Bound

Exact Result

Rewritten SQL

SQL DB

user/app

select avg(price) from sales
where channel = 'online'

select avg(price), std(price)
from (select avg(price) as a1
from sales
where channel = 'online'
group by sid) t1

Rewritten SQL

AQP

SQL

AQP

Approx Result

Exact Result

Rewritten SQL

SQL DB

user/app
Our proposal: Universal AQP

Approx Result + Error Bound

Exact Result

Rewritten SQL

select avg(price) from sales
where channel = 'online'

select avg(a1), std(a1) from (select avg(price) as a1 from sales
where channel = 'online'
group by sid) t1

Thin AQP layer

SQL DB

user/app
Our proposal: Universal AQP

Universal AQP

Approx Result + Error Bound

Thin AQP layer

Exact Result

Rewritten SQL

SQL DB

user/app

select avg(price) from sales
where channel = 'online'

Rewritten SQL

select avg(al), std(al)
from (select avg(price) as al
from sales
where channel = 'online'
group by sid ) t1
Challenges of Universal AQP

1. Statistical correctness (inter-tuple correlations)
   • Foreign-key constraints [Join Synopses ‘99]
   • Modifying the join algorithm [Wander Join ‘16]
   • Modifying the query plan [BlinkDB ‘13, Quickr ‘16]

2. Middleware efficiency
   • Lack of access to DBMS machinery

3. Server efficiency
   • Resampling-based techniques [Pol and Jermaine ‘05, BlinkDB ‘14]
   • Intimate integration of err est. logic into scan operators [Quickr ‘16, SnappyData]
   • Overriding the relational operators altogether [ABM ‘14]
Challenges of Universal AQP

1. **Statistical correctness (inter-tuple correlations)**
   - Foreign-key constraints [Join Synopses ‘99]
   - Modifying join algorithm [Wander Join ‘16]
   - Modifying the query plan [Quickr ‘16]

![Bar chart showing sales in AA, NYC, and SF with correct error bounds.](chart.png)
Challenges of Universal AQP

1. Statistical correctness (inter-tuple correlations)
   • Foreign-key constraints [Join Synopses ‘99]
   • Modifying join algorithm [Wander Join ‘16]
   • Modifying the query plan [Quickr ‘16]

2. Middleware efficiency
   • Lack of access to DBMS machinery
Challenges of Universal AQP

1. **Statistical correctness (inter-tuple correlations)**
   - Foreign-key constraints [Join Synopses ‘99]
   - Modifying join algorithm [Wander Join ‘16]
   - Modifying the query plan [Quickr ‘16]

2. **Middleware efficiency**
   - Lack of access to DBMS machinery

3. **Server efficiency**
   - Resampling-based techniques [Pol and Jermaine ‘05, BlinkDB ‘14]
   - Intimate integration of err est. logic into scan operators [Quickr ‘16, SnappyData]
   - Overriding the relational operators altogether [ABM ‘14]
VerdictDB Overview

First Universal AQP system
Deployment

user/app

JDBC, API call

VerdictDB

SQL DB
Deployment

user/app  \(\xrightarrow{\text{JDBC, API call}}\)  VerdictDB  \(\xleftarrow{\text{JDBC, spark.sql}}\)  SQL DB
Deployment

Stores (1) offline-created samples, and (2) VerdictDB-managed metadata
Deployment

Stores (1) offline-created samples, and (2) VerdictDB-managed metadata

The only requirements:
• create table as select ...
• rand(), agg(col) over (partition by ...)

user/app  
JDBC, API call  
VerdictDB  
JDBC, spark.sql  
SQL DB
Deployment

 Stores (1) offline-created samples, and (2) VerdictDB-managed metadata

The only requirements:
• create table as select ...
• rand(), agg(col) over (partition by ...)

supported by almost any SQL engines
Architecture

VerdictDB

- Query Parser
- Query Rewriter
- Answer Rewriter

DBMS Drivers
- Impala driver
- Hive driver
- Redshift driver

incoming query

approximate answer

SQL DB
Crucial component

1. Chooses an optimal set of samples
2. Scales values appropriately
3. Inserts an error estimation logic
Error estimation in VerdictDB
Error estimation in general

User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$
User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$

**Main question:** how close is $Q(S)$ to $Q(T)$?
User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$

**Main question:** how close is $Q(S)$ to $Q(T)$?

<table>
<thead>
<tr>
<th></th>
<th>Fast?</th>
<th>General?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed-form (CLT, Hoeffding, HT)</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Existing Resampling (subsampling, bootstrap)</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Ours (variational subsampling)</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
### Error estimation in general

User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$

**Main question:** how close is $Q(S)$ to $Q(T)$?

<table>
<thead>
<tr>
<th></th>
<th>Fast?</th>
<th>General?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed-form</td>
<td>YES</td>
<td>NO (no UDAs)</td>
</tr>
<tr>
<td>(CLT, Hoeffding, HT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existing Resampling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(subsampling, bootstrap)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>(variational subsampling)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Error estimation in general

User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$

**Main question:** how close is $Q(S)$ to $Q(T)$?

<table>
<thead>
<tr>
<th></th>
<th>Fast?</th>
<th>General?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed-form</td>
<td>YES</td>
<td>NO (no UDAs)</td>
</tr>
<tr>
<td>(CLT, Hoeffding, HT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Existing Resampling</td>
<td>NO (can be slow in SQL)</td>
<td>YES (Hadamard differentiable)</td>
</tr>
<tr>
<td>(subsampling, bootstrap)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>(variational subsampling)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Error estimation in general**

User interested in $Q(T)$

We compute $Q(S)$ where $S$ is a sample of $T$

**Main question:** how close is $Q(S)$ to $Q(T)$?

<table>
<thead>
<tr>
<th></th>
<th>Fast?</th>
<th>General?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Closed-form</strong></td>
<td><strong>YES</strong></td>
<td>NO (no UDAs)</td>
</tr>
<tr>
<td>(CLT, Hoeffding, HT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Existing Resampling</strong></td>
<td>NO (can be slow in SQL)</td>
<td><strong>YES</strong> (Hadamard differentiable)</td>
</tr>
<tr>
<td>(subsampling, bootstrap)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>YES</strong></td>
<td><strong>YES</strong> (Hadamard differentiable)</td>
</tr>
<tr>
<td>(variational subsampling)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recap: traditional subsampling

Original Table
(size N)

\( Q(T) \) is slow / expensive
Recap: traditional subsampling

What is the error of $Q(S)$?
Recap: traditional subsampling

What is the error of \( Q(S) \)?
Recap: traditional subsampling

- Random sample without replacement
- Each subsample is independent

Original Table (size $N$)

Subsample (size $s \ll n$)

What is the error of $Q(S)$?
Recap: traditional subsampling

What is the error of $Q(S)$?
Recap: traditional subsampling

What is the error of $Q(S)$?

Original Table (size $N$)

random sample

sample (size $n$)

subsample (size $s \ll n$)

$s_1$, $s_2$, $s_3$, ..., $s_b$

$Q(s_1)$, $Q(s_2)$, $Q(s_3)$, $Q(s_b)$
Recap: traditional subsampling

Important properties

1. A tuple may belong to multiple subsamples.
2. The size of every subsample is $s$. 

What is the error of $Q(S)$?
Traditional subsampling in SQL is slow

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>1</th>
<th>2</th>
<th>···</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$\text{sum} = s$
Traditional subsampling in SQL is slow

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm:
for $i = 1, \ldots, n$
for $j = 1, \ldots, b$
if $sid[i,j] == 1$
    $sum[j] += price[i]$
**Traditional subsampling in SQL is slow**

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm:

```plaintext
for i = 1, ..., n
    for j = 1, ..., b
        if sid[i,j] == 1
            sum[j] += price[i]
```

Time Complexity: $O(n \cdot b)$
Traditional subsampling in SQL is slow

Algorithm:
for $i = 1, \ldots, n$
   for $j = 1, \ldots, b$
      if $\text{sid}[i,j] == 1$
         $\text{sum}[j] += \text{price}[i]$

Time Complexity: $O(n \cdot b)$

No error est: 0.35 sec
Trad. subsampling: 118 sec
337x slower

(based on 1G sample, Impala)
Our approach: variational subsampling

T → random sample → S (sample, size n)
Our approach: variational subsampling
Our approach: variational subsampling

Important properties
1. A tuple may belong to multiple subsamples.
2. The size of every subsample is s.
Our approach: variational subsampling

Important properties
1. A tuple may belong to multiple subsamples.  
   *Each sampled tuple can belong to at most one subsample*
2. The size of every subsample is $s$. 

$T$ random sample

$S$ sample (size $n$)

$s_1$ (size $n_1$)  
$s_2$ (size $n_2$)  
$s_3$ (size $n_3$)  
$\cdots$  
$s_b$ (size $n_b$)
Our approach: variational subsampling

Important properties
1. A tuple may belong to multiple subsamples. Each sampled tuple can belong to at most one subsample.
2. The size of every subsample is $s$. Allow subsamples to differ in size.
Our approach: variational subsampling

Important properties
1. A tuple may belong to multiple subsamples. Each sampled tuple can belong to at most one subsample
2. The size of every subsample is $s$. Allow subsamples to differ in size.

Can be implemented in SQL as a single group-by query!
Variational subsampling in SQL is fast

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>subsample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>3</td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>2</td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>3</td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>5</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>5</td>
</tr>
</tbody>
</table>

We call this augmented table, a variational table.
Variational subsampling in SQL is fast

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>subsample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>3</td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>2</td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>3</td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>5</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>5</td>
</tr>
</tbody>
</table>

Algorithm:

for i = 1, ..., n
    sum[sid] += price[i]

We call this augmented table, a variational table
Variational subsampling in SQL is fast

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>subsample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>3</td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>2</td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>3</td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>5</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>5</td>
</tr>
</tbody>
</table>

Algorithm:
for i = 1, ..., n
    sum[sid] += price[i]

Time Complexity: $O(n)$

We call this augmented table, a variational table.
Variational subsampling in SQL is fast

<table>
<thead>
<tr>
<th>CITY</th>
<th>PRODUCT</th>
<th>PRICE</th>
<th>subsample ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>1</td>
</tr>
<tr>
<td>AA</td>
<td>milk</td>
<td>$5.00</td>
<td>3</td>
</tr>
<tr>
<td>AA</td>
<td>egg</td>
<td>$3.00</td>
<td>2</td>
</tr>
<tr>
<td>NYU</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>NYU</td>
<td>milk</td>
<td>$6.00</td>
<td>3</td>
</tr>
<tr>
<td>NYU</td>
<td>candy</td>
<td>$2.00</td>
<td>1</td>
</tr>
<tr>
<td>SF</td>
<td>milk</td>
<td>$6.00</td>
<td>5</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>4</td>
</tr>
<tr>
<td>SF</td>
<td>egg</td>
<td>$4.00</td>
<td>5</td>
</tr>
</tbody>
</table>

We call this augmented table, a variational table

Algorithm:

for i = 1, ..., n  
  sum[sid] += price[i]

Time Complexity: $O(n)$

No error est: 0.35 sec
Trad. subsampling: 118 sec
Var. subsampling: 0.73 sec

162× faster than traditional

(based on 1G sample, Impala)
Main results

Theorem 1 (Consistency) The distribution of the aggregates of variational subsamples, after appropriate scaling, converges to the true distribution of the aggregate of a sample as $n \to \infty$. 
**Main results**

**Theorem 1 (Consistency)** *The distribution* of the aggregates of *variational subsamples*, after appropriate scaling, *converges to the true distribution* of the aggregate of a sample as \( n \to \infty \).

**Theorem 2 (Convergence Rate)** The convergence rate of variational subsampling is equal to that of traditional subsampling *when b is finite*.

\[
O \left( n_s^{-1/2} + \frac{n_s}{n} + b^{-1/2} \right)
\]

The error term from the finite b
(The Dvoretzky–Kiefer–Wolfowitz inequality)
Experiments

1. Does VerdictDB provide enough speedup?
2. Is VerdictDB (UAQP)'s performance comparable to a tightly-integrated AQP?
3. Is variational subsampling statistically correct?
Experiments

1. Does VerdictDB provide enough speedup?
Experiments

1. Does VerdictDB provide enough speedup?

2. Is VerdictDB (UAQP)’s performance comparable to a tightly-integrated AQP engine?
Experiments

1. Does VerdictDB provide enough speedup?
2. Is VerdictDB (UAQP)’s performance comparable to a tightly-integrated AQP engine?
3. Is variational subsampling statistically correct?
Experiments

1. Does VerdictDB provide enough speedup?

2. Is VerdictDB (UAQP)’s performance comparable to a tightly-integrated AQP engine?

3. Is variational subsampling statistically correct?

Yes
Experiments

1. Does VerdictDB provide enough speedup?
2. Is VerdictDB (UAQP)’s performance comparable to a tightly-integrated AQP engine?
3. Is variational subsampling statistically correct?

Datasets:
- 500GB TPC-H benchmark / 200GB Instacart dataset / synthetic datasets

Underlying databases
- Amazon Redshift, Apache Spark SQL, Apache Impala on 10+1 r4.xlarge cluster
Speedup for Redshift

Redshift 24.0× Speedup

tpc-h benchmark

micro-benchmark
Speedup for Redshift

**tpc-h benchmark**

**micro-benchmark**

t3, t10, t15: no speedup (i.e., 1×) due to high-cardinality grouping attributes
Speedup for Redshift

**tpc-h benchmark**

- t3, t10, t15: no speedup (i.e., $1 \times$) due to high-cardinality grouping attributes

**micro-benchmark**

Other queries: **26.3× speedups** (relative errors were 2%)
Speedup for Apache Spark & Impala

- **Spark SQL**: 12.0× Speedup
- **Impala**: 18.6× Speedup
Speedup for Apache Spark & Impala

\[ \text{speedup} = \frac{\text{overhead + processing}}{\text{overhead} + (\text{sample processing})} \]

Lower overhead \( \rightarrow \) Larger speedup

Spark SQL: 12.0\times \text{Speedup}

Impala: 18.6\times \text{Speedup}
UAQP vs. Tightly-integrated AQP
VerdictDB was comparable to SnappyData.
UAQP vs. Tightly-integrated AQP

VerdictDB was comparable to SnappyData.

SnappyData ver 0.8 didn’t support the join of two sample tables.
Variational subsampling: correctness

Rel. err. naturally become smaller for higher selectivity.

The bars are 5th and 95th percentiles.

(a) Estimated error for different selectivity

(b) Estimated error for different sample sizes
Variational subsampling: correctness

Rel. err. naturally become smaller for higher selectivity.

(a) Estimated error for different selectivity

(b) Estimated error for different sample sizes

The estimated errors close to true errors.
Variational subsampling: correctness

Rel. err. naturally become smaller for higher selectivity.

(b) Estimated error for different sample sizes

The bars are 5th and 95th percentiles.

(a) Estimated error for different selectivity

The estimated errors close to true errors.

The accuracy of var. subsampling ≈ (a) bootstrap and (b) trad. subsampling
Variational subsampling: convergence rate

(a) Accuracy of error bound estimation

(b) Latency of error bound estimation
Variational subsampling: convergence rate

The accuracy was almost the same for relatively large samples.
Variational subsampling: convergence rate

Variational subsampling was significantly faster. The accuracy was almost the same for relatively large samples.
Conclusion: Universal AQP is viable
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution

2. New error estimation technique: *variational subsampling*
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution

2. New error estimation technique: *variational subsampling*
   
   1. Generality and computational efficiency
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution

2. New error estimation technique: variational subsampling
   1. Generality and computational efficiency
   2. The first subsampling-based error estimation technique for AQP
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution

2. New error estimation technique: variational subsampling
   1. Generality and computational efficiency
   2. The first subsampling-based error estimation technique for AQP

3. Offers considerable speedup
   (18.45× on average, up to 171×, less than 2-3% errors)
Conclusion: Universal AQP is viable

1. Comparable performance to a fully-integrated solution

2. New error estimation technique: variational subsampling
   1. Generality and computational efficiency
   2. The first subsampling-based error estimation technique for AQP

3. Offers considerable speedup
   (18.45× on average, up to 171×, less than 2-3% errors)

Open-sourced (Apache v2.0): http://verdictdb.org
Future Work

Development

Research
Future Work

Development

• Adding more drivers (Presto, Teradata, Oracle, SQL Server, ...)

Research
Future Work

Development

• Adding more drivers (Presto, Teradata, Oracle, SQL Server, …)

Research

• Support for online sampling
Future Work

Development

• Adding more drivers (Presto, Teradata, Oracle, SQL Server, ...)

Research

• Support for online sampling

• Robust physical designer (see CliffGuard @ SIGMOD 15)
Future Work

Development

• Adding more drivers (Presto, Teradata, Oracle, SQL Server, ...)

Research

• Support for online sampling

• Robust physical designer (see CliffGuard @ SIGMOD 15)

• Integration with ML libraries (sampling-based model tuning)
Thank You
VerdictDB: current status

• We support
  • aggregates: sum, count, avg, count-distinct, quantiles, UDAs
  • sources: base table, derived table, equi-join
  • filters: comparison, some subquery
  • others: group-by, having, etc.

• Open-sourced under Apache License version 2.0
  • http://verdictdb.org for code and documentation

• Upcoming features
  • Online sampling, automated physical designer
VerdictDB: current status

• We support
  • aggregates: sum, count, avg, count-distinct, quantiles, UDAs
  • sources: base table, derived table, equi-join
  • filters: comparison, some subquery
  • others: group-by, having, etc.

• Open-sourced under Apache License version 2.0
  • http://verdictdb.org for code and documentation
VerdictDB: current status

• We support
  • aggregates: sum, count, avg, count-distinct, quantiles, UDAs
  • sources: base table, derived table, equi-join
  • filters: comparison, some subquery
  • others: group-by, having, etc.

• Open-sourced under Apache License version 2.0
  • http://verdictdb.org for code and documentation

• Upcoming features
  • Online sampling, automated physical designer
Example of query rewriting

**original**
```
select l_returnflag, count(*) as cc
from lineitem
group by l_returnflag;
```

**rewritten**
```
select vt1.`l_returnflag` AS `l_returnflag`,
    round (sum ((vt1.`cc` * vt1.`sub_size`)) / sum (vt1.`sub_size`)) AS `cc`,
    (stddev(vt1.`count_order`) * sqrt(avg(vt1.`sub_size`)))
    / sqrt(sum(vt1.`sub_size`)) AS `cc_err`
from (select vt0.`l_returnflag` AS `l_returnflag`,
    ((sum((1.0 / vt0.`sampling_prob`)) / count(*))
    * sum(count(*)) OVER (partition BY vt0.`l_returnflag`) AS `cc`,
    vt0.`sid` AS `sid`, count(*) AS `sub_size`
    from lineitem_sample vt0
    GROUP BY vt0.`l_returnflag`, vt0.`sid`) AS vt1
GROUP BY vt1.`l_returnflag`;
```


Variational subsampling: overhead
Variational subsampling: overhead

Overhead of variational subsampling: 0.38–0.87 seconds
Variational subsampling was $100 \times -237 \times$ faster compared to Consolidated Bootstrap.

Overhead of variational subsampling: $0.38 - 0.87$ seconds