AQP++: Connecting *Approximate Query Processing* with *Aggregate Precomputation* for Interactive Analytics

Jinglin Peng, Dongxiang Zhang, Jiannan Wang, Jian Pei

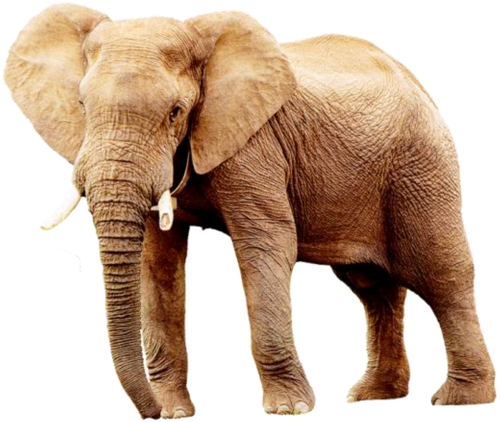
Simon Fraser University

06/14/2018

Interactive Analytics

Hardware

Big Data



How to enable **interactive** analytics over big data?

Two Separate Ideas

1. **A**pproximate **Q**uery **P**rocessing (**AQP**)

2. **Agg**regate **Pre**computation(**AggPre**)

Running Example

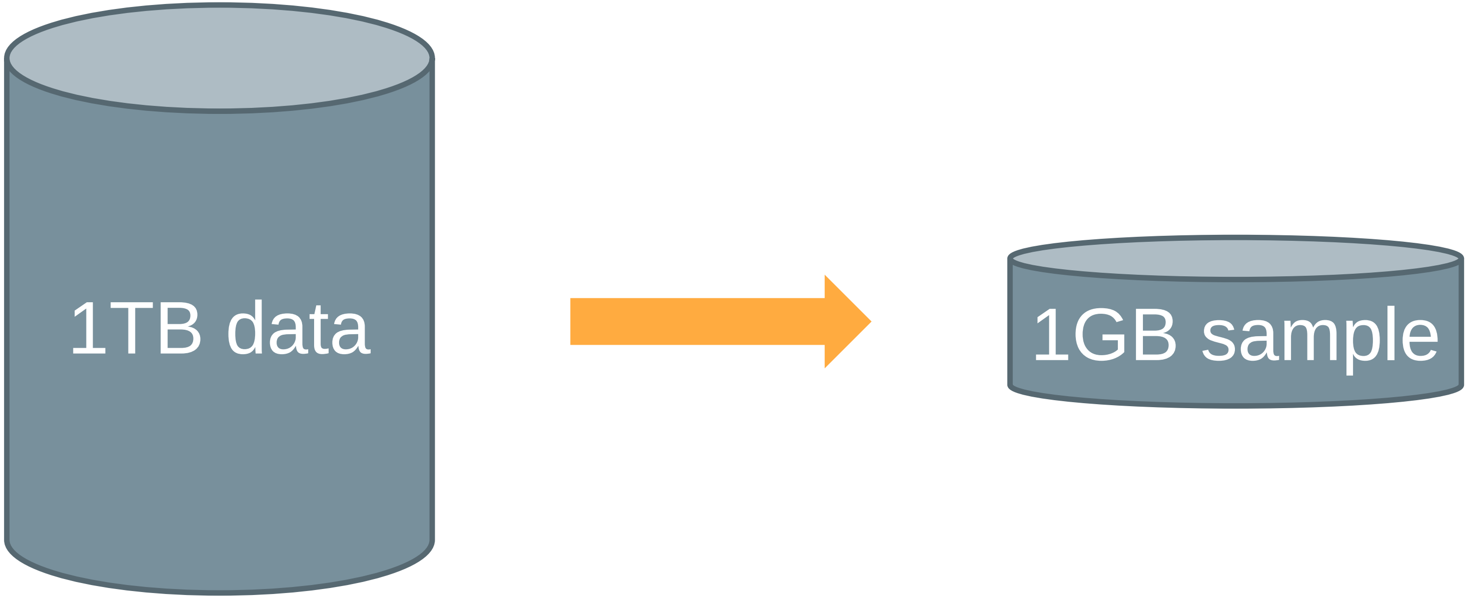
query: SELECT SUM(price) WHERE id in [1000, 9000]

Fact Table

ID	Price
1	100
2	50
…	
1000	900
…	
9000	70
…	
12000	500

Idea 1: AQP

SELECT SUM(price) WHERE id in [1000, 9000]



Idea 2: AggPre

SELECT SUM(price) WHERE id in [1000, 9000]

Prefix-Sum Cube[1]

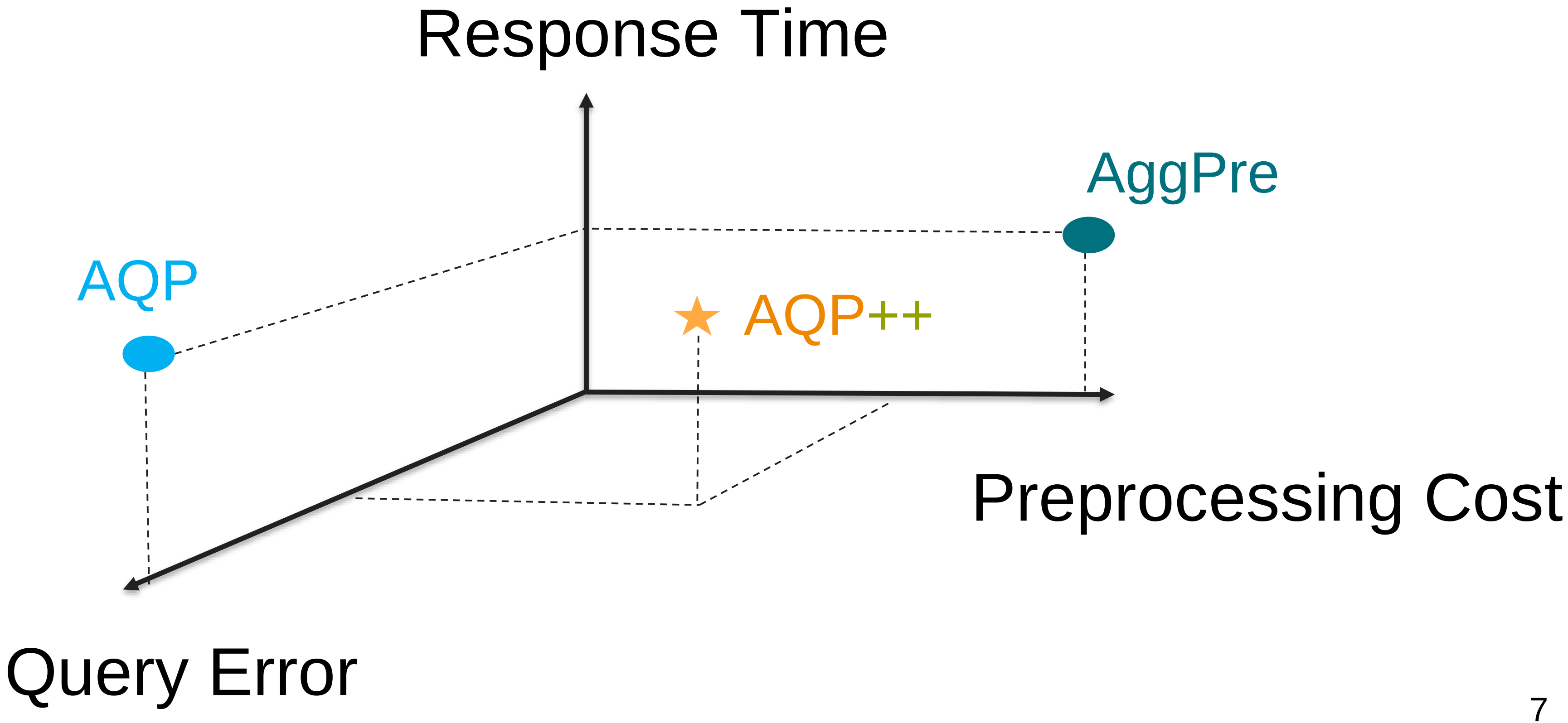
ID	Price
≤1	100
≤2	150
…	
≤999	$1.1*10^5$
…	
≤9000	$8.1*10^5$
…	
≤12000	$9.8*10^5$

SELECT SUM(price)
WHERE **id ≤ 999**

SELECT SUM(price)
WHERE **id ≤ 9000**

[1] Ho, Ching-Tien, et al. Range queries in OLAP data cubes. (1997)

Tradeoff


Response Time
AggPre
AQP
★ AQP++
Preprocessing Cost
Query Error

How AQP++ works?

SELECT SUM(Price) WHERE id in [1000, 9000]

AQP++ estimator

SELECT SUM(Price)
WHERE id in (1100,9000]

SELECT SUM(Price)
WHERE id in [1000, 1100]

Partial Cube

ID	Price
≤500	$5.1*10^4$
≤1100	$9.7*10^4$
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤9000	$8.1*10^5$
≤12000	$9.8*10^5$

1GB Sample

AQP++ Estimator

AQP

q: SELECT f(A) FROM D
WHERE condition_1

$$\text{Est} = \hat{q}(S)$$

sample

AQP++

q: SELECT f(A) FROM D
WHERE condition_1

pre: SELECT f(A) FROM D
WHERE condition_2

$$\text{Est} = \text{pre(D)} + \{\hat{q}(S) - \text{p}\hat{re}(S)\}$$

cube

sample

Good Properties of AQP++

1. Unify AQP and AggPre.

2. Support any aggregate function that AQP can support (e.g., SUM, COUNT, AVG, VARIANCE).

3. Easy to implement.

Research Challenges

- Query Processing

- Preprocessing

Query Processing Challenge

SELECT SUM(Salary) WHERE id in [1000, 8000]

ID	Price
≤500	$5.1*10^4$
≤1100	$9.7*10^4$
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤9000	$8.1*10^5$
≤12000	$9.8*10^5$

(500,1100]
(500,2000]
(500,4000]
(500,9000]
(500,12000]
(1100,2000]
(1100,4000]
(1100,9000]
(1100,12000]
(2000,4000]
(2000,9000]
(2000,12000]
(4000,9000]
(4000,12000]
(9000,12000]
(-∞,500]
(-∞,1100]
(-∞,2000]
(-∞,4000]
(-∞,9000]
(-∞,12000]

Too many precomputed queries!

How to **efficiently** find the best one?

Step1: Get Candidates

SELECT SUM(Salary) WHERE id in [1000, 8000]

ID	Price
≤500	$5.1*10^4$
≤1100	$9.7*10^4$
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤9000	$8.1*10^5$
≤12000	$9.8*10^5$

Candidates

pre 1: (500, 4000]

pre 2: (500, 9000]

pre 3: (1100, 4000]

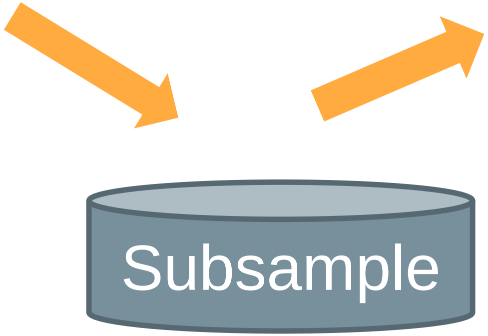
pre 4: (1100, 9000]

pre 5: Ø

Step 2: Estimate Errors

SELECT SUM(Salary) WHERE id in [1000, 8000]

pre 1: (500, 4000]
pre 2: (500, 9000]
pre 3: (1100, 4000]
pre 4: (1100, 9000]
pre 5: Ø



Err(pre 1): 2.5%
Err(pre 2): 1.2%
Err(pre 3): 2%
Err(pre 4): 1%
Err(pre 5): 10%

Research Challenges

- Query Processing

- Preprocessing

Preprocessing Challenge

Partial Cube 1

ID	Price
≤500	$5.1*10^4$
≤1100	$9.7*10^4$
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤9000	$8.1*10^5$
≤12000	$9.8*10^5$

Partial Cube 2

ID	Price
≤1	100
≤2	150
≤3	1150
≤4	1200
≤5	1330
≤6	1600

Partial Cube 3

ID	Price
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤6000	$6.3*10^5$
≤8000	$7.5*10^5$
≤10000	$8.9*10^5$
≤12000	$9.8*10^5$

……

Too many possible cubes!

Given a space budget, how to find the best cube?

Theoretical Result

ID	Price
≤2000	$2.5*10^5$
≤4000	$4.6*10^5$
≤6000	$6.3*10^5$
≤8000	$7.5*10^5$
≤10000	$8.9*10^5$
≤12000	$9.8*10^5$

Equal partition is optimal when:

1. Condition column has no duplicate.
2. Aggregate column and condition column are independent.

Hill Climbing for General Case

Basic Idea

- Initialization: Equal partition.
- Adjustment: Move one partition point to form a better cube.

Two Issues

1. How to efficiently compute the error of cube?

2. Which point & where to move?

Supported Queries

SELECT SUM(A)
FROM table
WHERE C

Supported Queries

SELECT f(A)
FROM table
WHERE C

f=SUM, COUNT, AVG, VARIANCE,…

Supported Queries

SELECT f(A)
FROM table
WHERE $C_1, C_2, \ldots C_n$

f=SUM, COUNT, AVG, VARIANCE,…

Supported Queries

SELECT f(A)
FROM table
WHERE $C_1, C_2, \dots C_n$
GROUP-BY G

f=SUM, COUNT, AVG, VARIANCE,…

Supported Queries

SELECT f(A)
FROM $table_1$, $table_2$
WHERE **$table_1$.PK = $table_2$.FK** & C_1, C_2, … C_n
GROUP-BY G

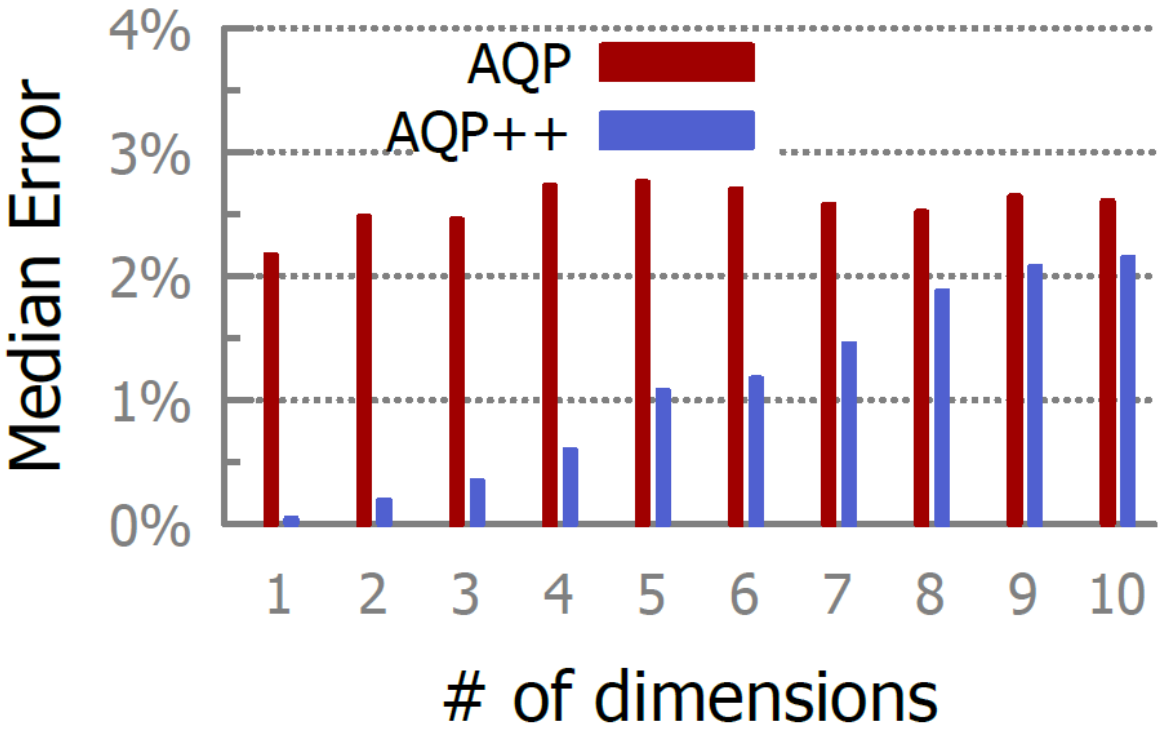
f=SUM, COUNT, AVG, VARIANCE,…

Exp: Setup

- Environment
  - Windows
  - 16 GB RAM, 1TB HDD
  - Visual Studio C++
- Dataset
  - TPCD benchmark (100GB, 600 million, skewness=2)
  - BigData benchmark (100GB, 752 million)
  - TLCTrip (200GB, 1400 million)
- Default Parameter
  - Sample ratio: 0.05%
  - Cube size: 50000
  - Query selectivity: 0.5% to 5%

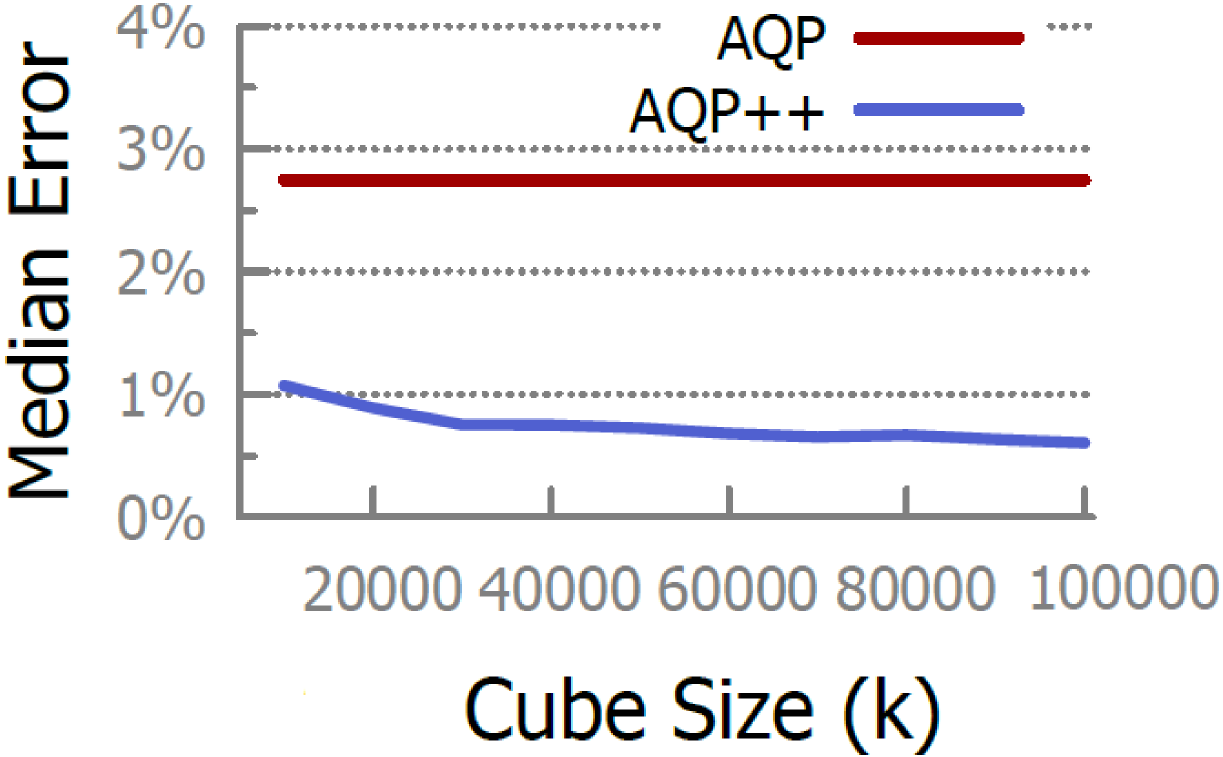
AQP++ can scale up to 10 dims

TPCD benchmark



A small cube can improve quality a lot

BigData benchmark



Performance Summary

TPCD benchmark
2-dim Query

	Preprocessing Cost		Response Time	Average Error
	Space	Time		
AQP	51.2 MB	4.3 min	0.60 sec	2.67%
AggPre	> 10 TB	> 1 day	< 0.01 sec	0.00%
AQP++	51.9 MB	11.7 min	0.67 sec	0.27%

AQP++ can be up to 10X more accurate!

Related Work

- Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. **"Revisiting reuse for approximate query processing."** VLDB’2017
- Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, and Barzan Mozafari. **"Database learning: Toward a database that becomes smarter every time. "** SIGMOD’2017
- Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. **"Sample+ seek: Approximating aggregates with distribution precision guarantee."** SIGMOD’2016
- Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. **"A sample-and-clean framework for fast and accurate query processing on dirty data."** SIGMOD’2014
- Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. **"Distributed and interactive cube exploration."** ICDE’2014
- Ruoming Jin, Leonid Glimcher, Chris Jermaine, and Gagan Agrawal. **"New sampling-based estimators for OLAP queries."** *ICDE’2006*

Key Takeaways

SFU-DB
Database System Lab

- **AQP++:** Connect AQP with Aggregate Precomputation
- Achieve **a better tradeoff** among preprocessing cost, response time, and query quality.
- Up to **10x more accurate** than AQP.

https://github.com/sfu-db/aqppp

Thank You!

Q&A