AQP++

Connecting Approximate Query Processing With Aggregate Precomputation

Rohit Mittal INFO 290T

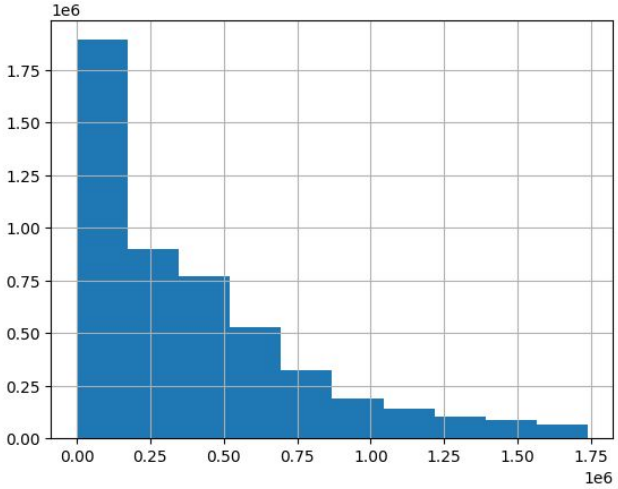
Problem - Interactivity

- Queries take time to execute
- Long running queries can break the interactivity of a dataset
- >5 second query latency = loss of focus

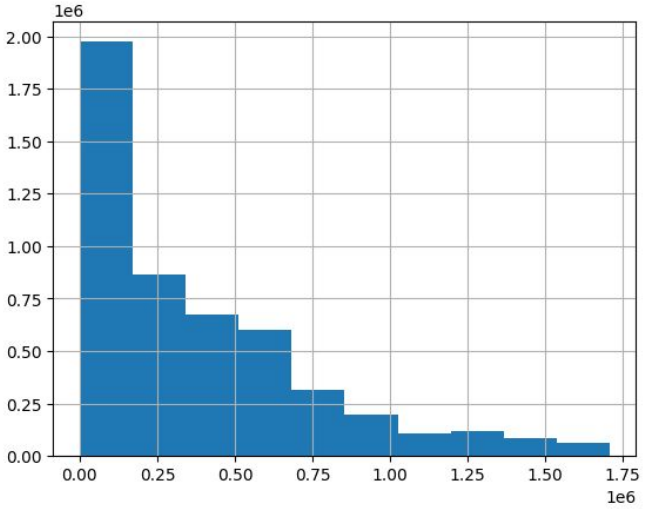
Background - AQP

- Approximate Query Processing
- Instead of scanning all the data associated with a query, generate an approximate answer based on a subset of the data
- Helps preserve interactivity
- Generally uses a random/stratified sample of data
- Trades off accuracy for increased speed
- Bootstraps a confidence interval
- Related in-class papers: BlinkDB, Pangloss, Sample+Seek

Histogram Query over 5M rows

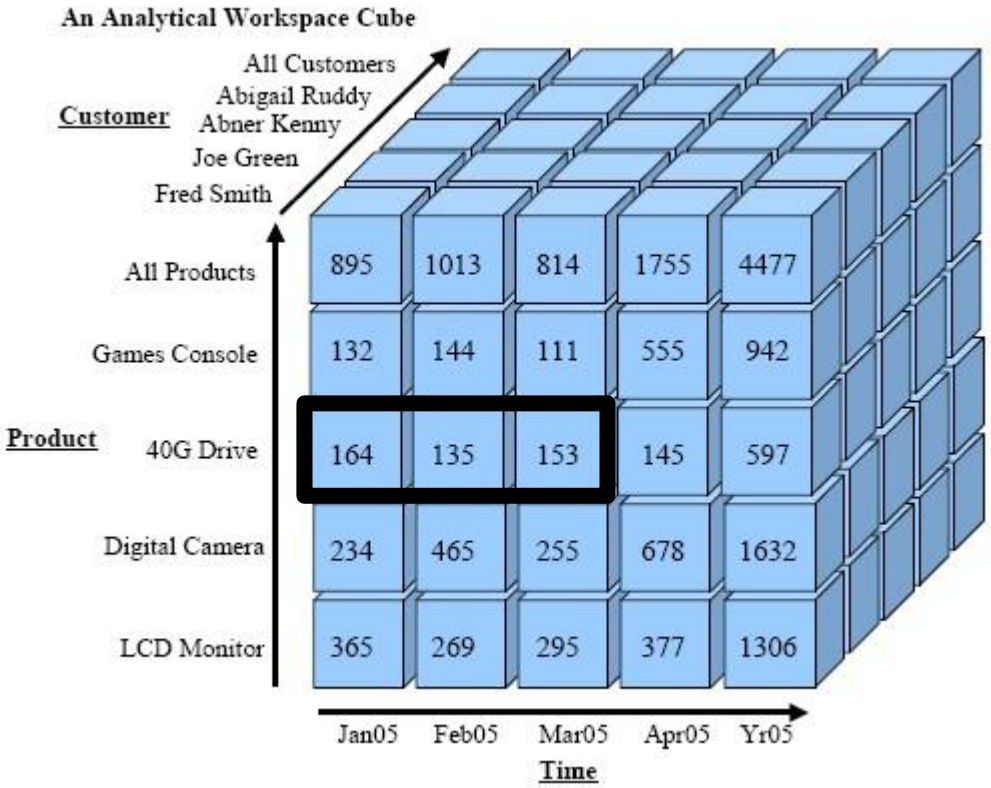


Approximate Query over 10k rows



Background - Aggressive Precomputation

- Precompute data in anticipation of queries
- Typically stored in data cubes
- Requires large amounts of computation time
- Requires large amounts of storage
- Hard to predict what to precompute



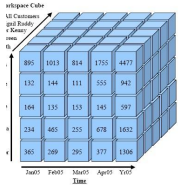
Source: http://aampbis.com/Images/Olap_Cube.jpg

```
SELECT COUNT(*)
FROM WORKSPACE
WHERE CUSTOMER = “Fred Smith”
AND PRODUCT = “40G Drive”
AND TIME >= JAN05
AND TIME <= MAR05
```

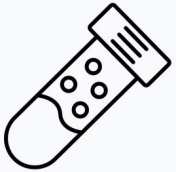
Solution - AQP++

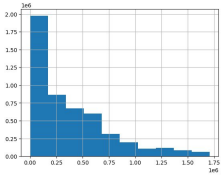
- Combine AQP with aggressive precomputation to deliver higher quality approximations
- Allows connecting AQP and its high interactivity with already existing precomputations in data warehouses

User Query



Existing Precomputation + Approximation of delta

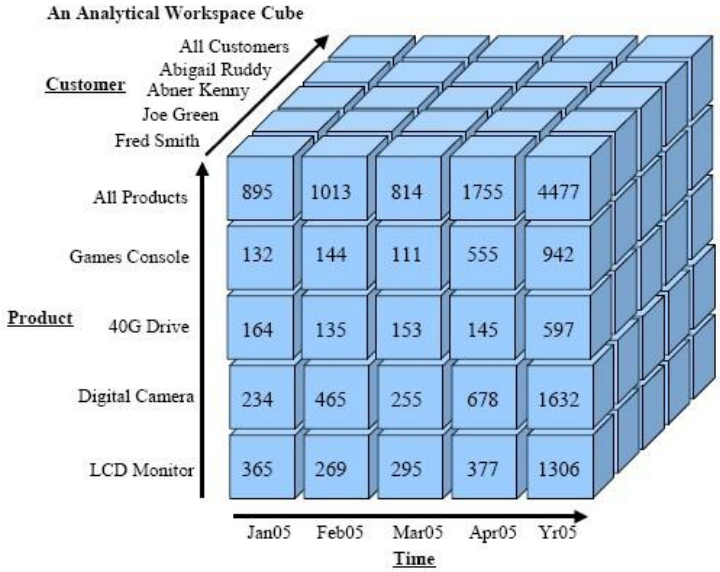




Output visualization that can only be more accurate than just using AQP!

But how do we know what to precompute?

- Can't pre-compute *everything*
- Query space is large - brute forcing pre-computations is impractical
- User specifies a *query template* of the aggregation they intend to use on the dataset and what conditional variables exist



Prefix Cube

- Each element of the data cube stores a prefix of the aggregation for the preceding elements in its domain
- Lower complexity even more with a blocked prefix cube
- Whiteboard time

Blocked Prefix Cube

- Blocked prefix cubes lower complexity by lowering the amount of elements per axis
- But how do we figure out where our blocks should be?
  - Uniform blocks are not optimal. More blocks should be allocated to the “interesting” parts of the data
  - Not all dimensions should have the same number of blocks

Binary Search

- AQP++ generates *error profiles* that measure how changing the number of possible blocks per dimension changes the possible query template error
- Uses binary search on those error profiles to figure out how many blocks it should allocate to each dimension to minimize error within the space budget provided

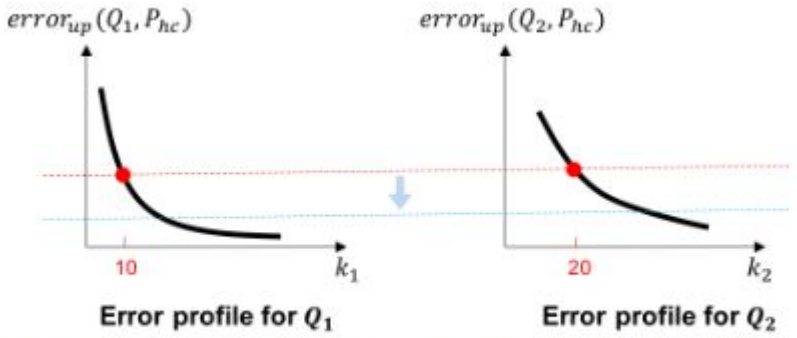


Figure 6: An illustration of the binary search algorithm to search for the BP-Cube's shape $k_1 \times k_2$ (suppose $k = 500$).

Hill Climbing

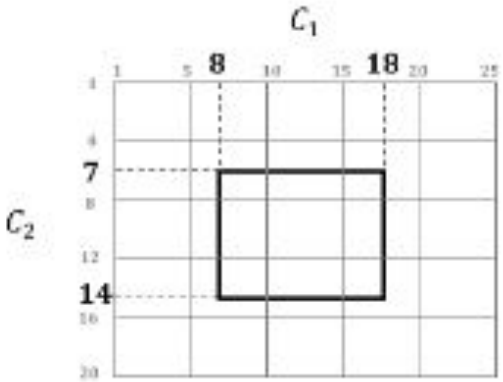
- AQP++ uses hill climbing to figure out where to partition blocks for each dimension
- Modifies one partition on an initial, uniform cube to see if that decreases error
- Error is calculated by finding the upper bound of query template error
  - Can be found in linear time!
- Stops when no adjustment from the current cube will decrease error

Now that we have these precomputations…

…how do we use them with queries?

Aggregate Identification

- Identify *candidate aggregate values*, blocks that contain some part of the query space we want

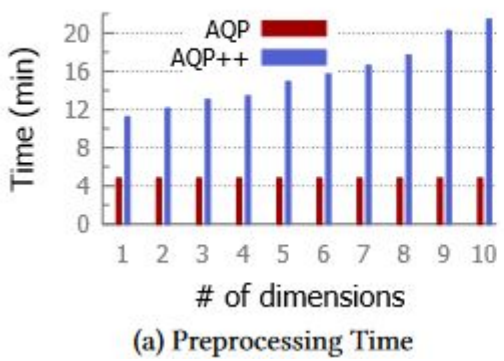


$$P^- = \{\ \text{SUM}(6{:}15, 5{:}12),\ \text{SUM}(11{:}15, 5{:}12),\ \text{SUM}(6{:}20, 5{:}12),\ \text{SUM}(11{:}20, 5{:}12),\ \text{SUM}(6{:}15, 9{:}12),\ \text{SUM}(11{:}15, 9{:}12),\ \text{SUM}(6{:}20, 9{:}12),\ \text{SUM}(11{:}20, 9{:}12),\ \text{SUM}(6{:}15, 5{:}16),\ \text{SUM}(11{:}15, 5{:}16),\ \text{SUM}(6{:}20, 5{:}16),\ \text{SUM}(11{:}20, 5{:}16),\ \text{SUM}(6{:}15, 9{:}16),\ \text{SUM}(11{:}15, 9{:}16),\ \text{SUM}(6{:}20, 9{:}16),\ \text{SUM}(11{:}20, 9{:}16),\ \emptyset\}$$

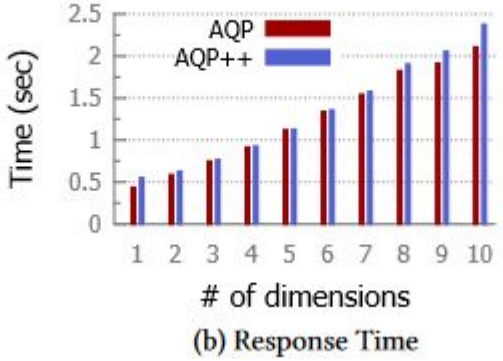
Aggregate Identification

- Use AQP to compute the user's query $Q(S)$
- For each candidate precomputation $Pre(D)$:
  - Compute what AQP would output if used to do the precomputation $\hat{p}(S)$
  - Return a candidate query result in the form $Pre(D) + \underbrace{(Q(S) - \hat{p}(S))}_{\text{Delta}}$
- Pick the query result with the best confidence interval
  - Calculated from subsamples of the AQP sample
- Edge cases
  - If no precomputations can possibly help, there will be no candidates and AQP++ will fall back to just AQP
  - If there is no delta, then the precomputation exactly matches the user query and an exact result can be returned.

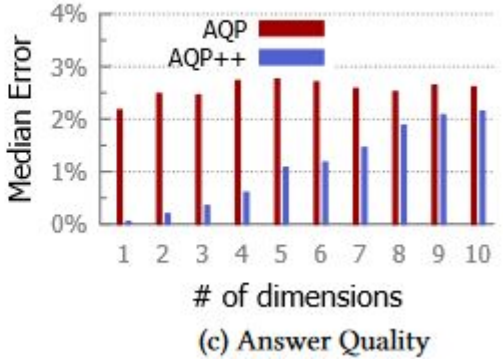
Impact



(a) Preprocessing Time



(b) Response Time



(c) Answer Quality

Limitations and Extensions

- More supported aggregations (ie. how do you combine the medians of 2 blocks?)
- Group-by queries (essentially treated as axes)
- How to efficiently handle updates
- How to handle space needs for sampling and BP-cubes
- How to handle space needs for multiple query templates

Questions?