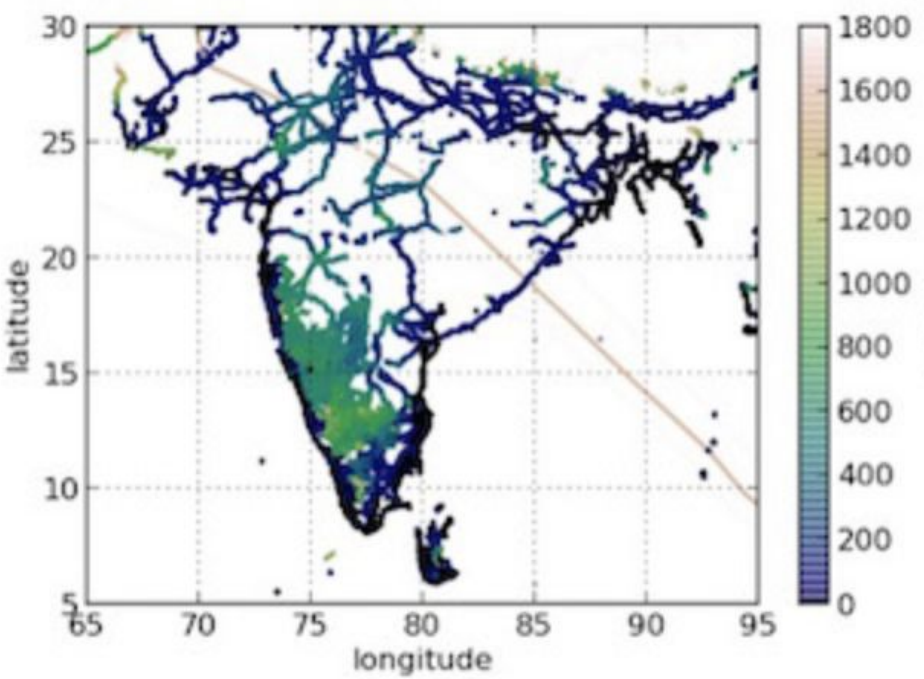
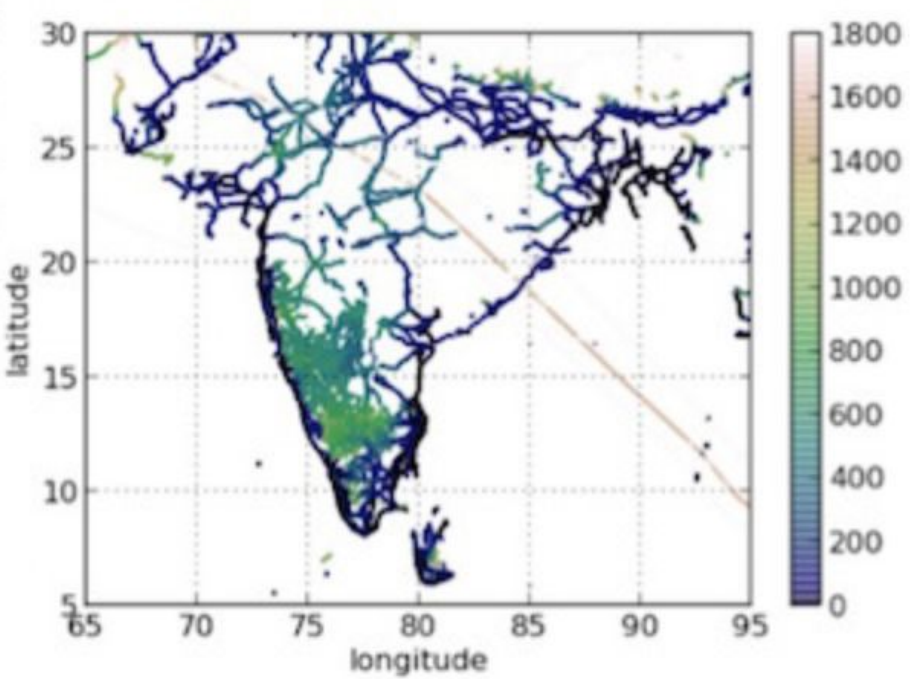
Quickr

Lazily Approximating Complex Adhoc Queries in Big Data Clusters

Andrew Levin, William Schmitt, Zhao Zhang


30
25
20
15
10
5
latitude
65
70
75
80
85
90
95
longitude
1800
1600
1400
1200
1000
800
600
400
200
0

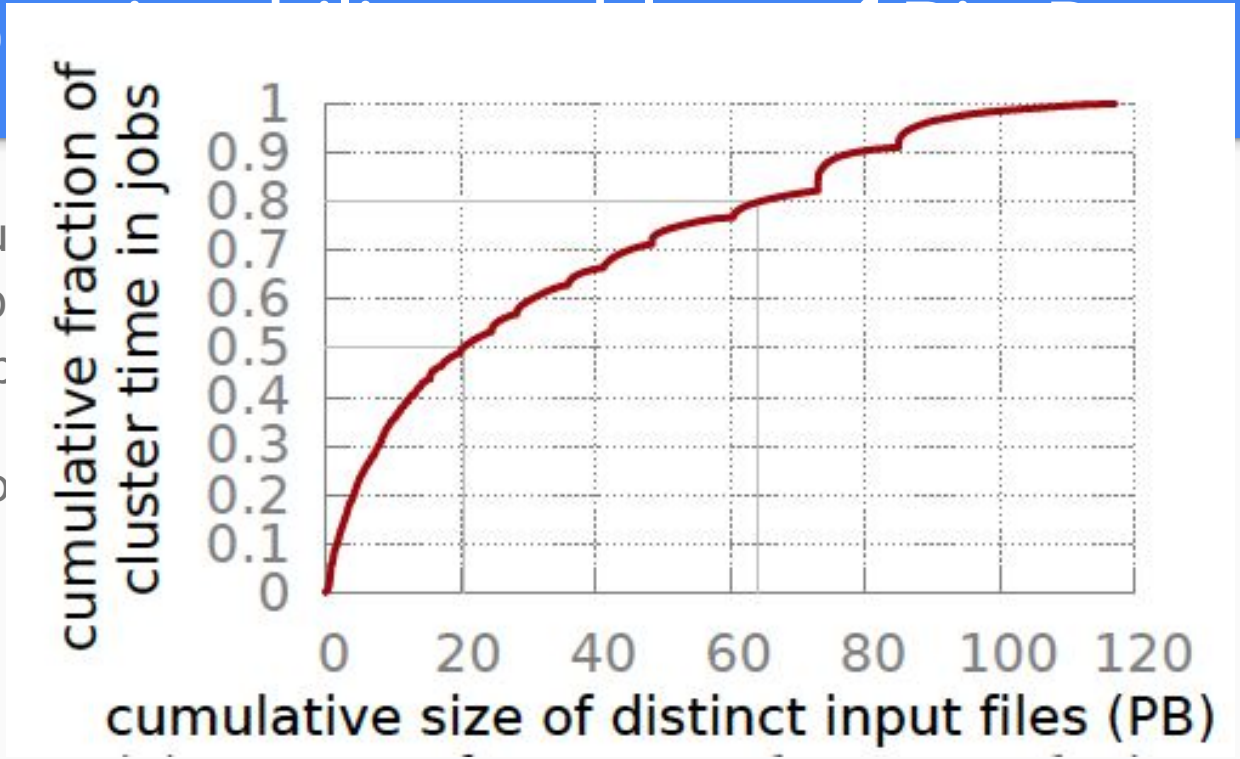

30
25
20
15
10
5
latitude
65
70
75
80
85
90
95
longitude
1800
1600
1400
1200
1000
800
600
400
200
0

Ideal System Goals

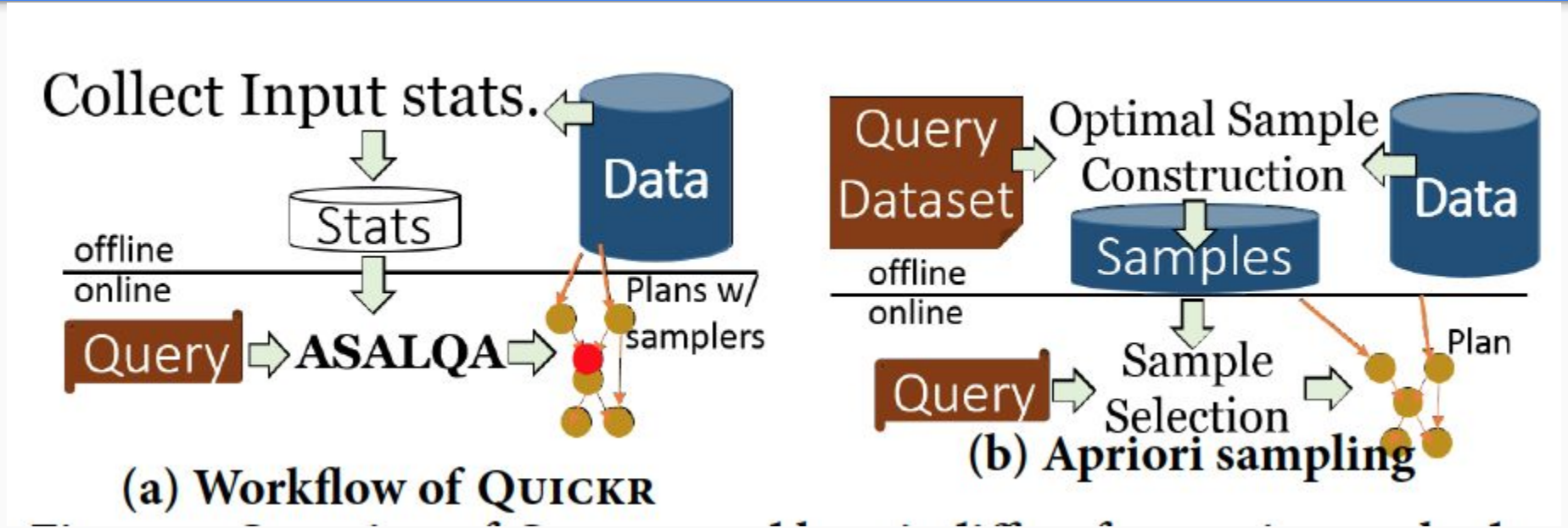
- Do not assume that input samples exist or that future queries are known
  - Just-In-Time sampling
- Offer turn-key support for approximations
- Support complex queries
- Ensure that answer will be accurate

App Queries

- Qu
- Ap
- Inp
  -
- No



Solution: QUICKR



Just-In-Time Sampling

In case of emergency: goo.gl/wAmusY

Samplers

3 Types of Samplers in Quickr:

- **Uniform** Sampler
- **Distinct** Sampler
- **Universe** Sampler

Each must be able run in:

- **Partitionable** mode
- **Streaming** mode
- A **single pass**

Samplers: Uniform Sampler

Details	● Filter that randomly allows each row through with uniform probability ● Use with reservoir sampling for best performance
Advantages	● Fast ● Meets targeted memory consumption
Disadvantage	● Misses small groups, skewed data

Samplers: Distinct Sampler

Details:	• Ensures that a (predetermined) number of distinct rows for each column set pass through filter • After the predetermined number of rows is let through, adds new rows with uniform probability
Advantage	• Captures small subgroups, **handles skewed data well**
Disadvantages	• **Biased** toward early rows in input • Very **high memory cost** • **Difficult to run in parallel**

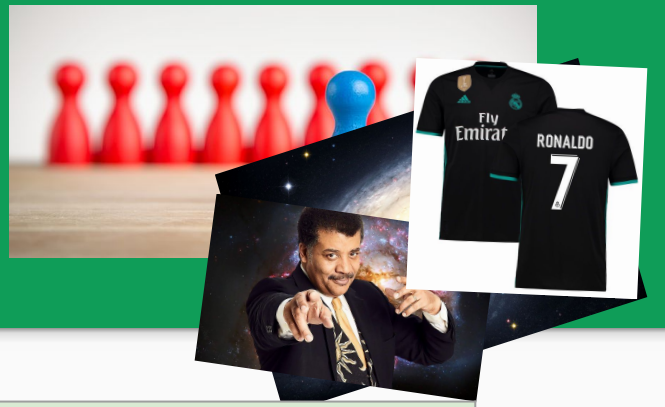
Samplers: Distinct Sampler

Disadvantages	• **Biased** toward early rows in input • Very **high memory cost** • **Difficult to run in parallel**
Improvements in Quickr	• Allow each instance to collect a fraction of the minimum number of rows (with a slack paramter) → now partitionable # handles most cases • Use approx freq of values in order to drop common values and only pick up values that skew • Periodically, randomly flush some rows and reweight reweight the remaining rows

Samplers: Universe Sampler

Details:	• Sample after join by mapping all rows to higher dimensional space using a strong hashing function
Advantage	• Fast way to sample across joins • Sample without apriori knowledge of the range of values in hash range • Only one pass, easily partitionable, keeps no state across rows • Works over arbitrarily many columns
Disadvantage	• More expensive than uniform sampling

Sampler Interaction



Pros	• Samplers compliment each other and can be used widely
Cons	• A new method to analyze accuracy because samplers are not uniformly random *(explained later)*
Notes	• Uniform, universe used only when there is no worry of missing subgroups • Universe can be used over multiple joins • Using samplers together allows otherwise unapproximable queries to benefit from Quickr

ASALQA

aahss - el - kaa

Approximate
Samplers inserted at
Appropriate
Locations in the
Query plan
Automatically

ASALQA = Samplers + Query Optimization

Quickr: add samplers to query optimization plan to speed up results

Challenges for ASALQA:

- Any operator can be followed by any sampler (leads to massive search space)
- Must reason about performance and accuracy for each location
- Must output most performant plan given target accuracy

Implementing ASALQA

Choice A:

1. Feed query to traditional query optimizer
2. Insert samplers into the optimized plan

Choice B:

- Use samplers as first-class operators along with other operators
- Explore larger combined space of possible plans with custom query optimizer

Normal Query Optimization

QO in Cascades has two main phases:

1. Logical exploration phase → generate alternative plans
2. Physical plan creation phase → convert each logical operation to a physical implementation

Implementing ASALQA

ASALQA modifies these slightly

- Samplers are injected into the query execution tree before every aggregation
- Introduces new transformation rules
- Use extra stats for dataset to identify best plan (for performance and accuracy)
    - Eg. frequency tables, heavy hitter analysis

Sampler Implementation (State)

- Sampler requirements are encoded during the logical exploration phase
  - Can change during transformation
- Logical state represented by: {*S*, *U*, *ds*, *sfm*}
  - *S* → columns to stratify on
  - *U* → columns to universe sample on
  - *ds* → downstream selectivity
  - *sfm* → stratification frequency multiplier

Sampler Implementation (Seeding)

Seed samplers by replacing each aggregation statement with:

- **precursor** statement ← JOIN, WHERE, UDOs, etc.
- **sampler** statement ← (U = ∅;ds = 1;sfm = 1)
- **successor** statement → performs aggregations on sampled columns

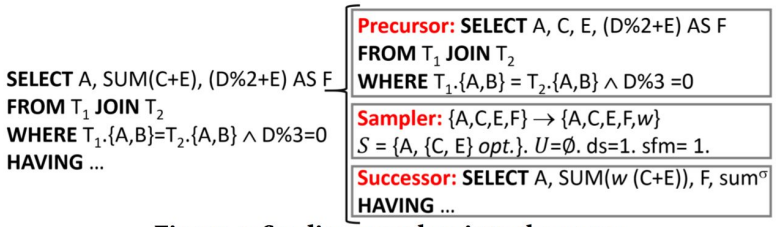


Figure 4: Seeding samplers into the query.

Definition: Query Transformation

The optimizer determines whether it is helpful to change the form of the query so that the optimizer can generate a better execution plan.

Eg. Transforming a query containing OR statements to use UNION ALL

```
SELECT *
FROM   sales
WHERE  promo_id=33
OR     prod_id=136;
```

→

```
SELECT *
FROM   sales
WHERE  prod_id=136
UNION ALL
SELECT *
FROM   sales
WHERE  promo_id=33
AND    LNNVL(prod_id=136);
```

From: docs.oracle.com/database/121/TGSQL/tgsql_transform.html

Pushing Samplers Past SELECT/JOIN

High level: given a statement, generate a series of possible transformations, pick the best option with at least better properties than the original.

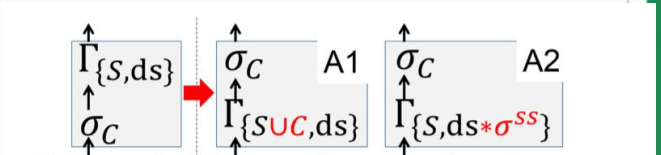


Figure 5: Pushing sampler past select: See §4.2.3.

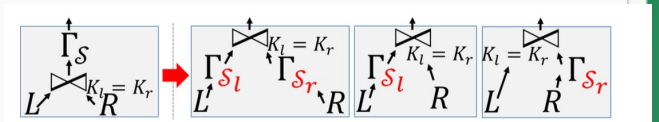


Figure 6: Pushing sampler past join: See §4.2.4.

Costing Sampled Expressions

Calculating cost:

- Key inputs to cost: cardinality, number of distinct values
- When to use each sampler:
  - Uniform → if *S* or *U* are empty or if there is substantial support
  - Universe → (has high variance) only chosen when needed and stratification needs are met
  - (No sampler) → per group support is small, distinct sampling would become too inefficient
  - Distinct → All other cases
- Physical parameters for each chosen to be smallest that ensure no subgroups are left out

Accuracy Analysis

Quickr provides:

- Unbiased estimators of aggregate values → Horvitz-Thompson estimator
- probability of missing groups
- Confidence intervals → using central limit theorem

Accuracy Analysis: Dominance

Problem: many different samplers, pushing samplers past operators introduces variance, bias

Solution: Dominance: $E_1 \Rightarrow E_2$ iff the accuracy of E_2 is no worse than that of E_1

Key Idea: use dominance to allow us to inductively determine inaccuracy of a query plan

Evaluation

Evaluation questions

- How much faster?
- How often correct?
- Where do gains come from?

Comparables

- Quickr vs two state-of-the-art systems
- Production QO w/o samplers is “baseline”
  - Thousands of man hours to develop, hardened from use
  - Supports almost all of T-SQL
  - Support for user defined objects (UDOs)
  - Generates parallel plans
  - Parallel implementation of operators
- BlinkDB
  - Given perfect matching

Query sets - TPC-DS benchmark

- Simpler than Microsoft queries
  - Fewer passes over data
  - Fewer joins

Metric	Percentile value				
	25th	50th	75th	90th	95th
# of Passes over Data	1.83	2.45	3.63	6.49	9.78
1/firstpass duration fract.	1.37	1.61	2.09	6.38	17.34
# operators	143	192	581	1103	1283
depth of operators	21	28	40	51	75
# Aggregation Ops.	2	3	9	37	112
# Joins	2	3	5	11	27
# user-defined aggs.	0	0	1	3	5
# user-defined functions	7	27	45	127	260
size of QCS+QVS	4	8	24	49	104

(b) Characterizing some $O(10^8)$ queries that ran in a production big-data cluster over a two month span.

Metric	Percentile value					
	10th	25th	50th	75th	90th	95th
# of passes	1.12	1.18	1.3	1.53	1.92	2.61
Total/First pass time	1.26	1.44	1.67	2	2.63	3.42
# Aggregation Ops.	1	1	3	4	8	16
# Joins	2	3	4	7	9	10
depth of operators	17	18	20	23	26	27
# operators	20	23	32	44	52	86
size of QCS + QVS	2	4	5	7	12	17
size of QCS	0	1	3	5	9	11
# user-defined func.	1	2	4	9	14	24

Table 3: Characteristics of the TPC-DS queries used in evaluation.

Performance metrics

- Machine-hours: sum of runtime of all tasks
  - Cluster occupancy
  - Throughput
- Intermediate Data: sum of the output of all tasks minus job output
  - Intermediate data written
- Shuffled Data
  - Sum of data that moves along network across racks

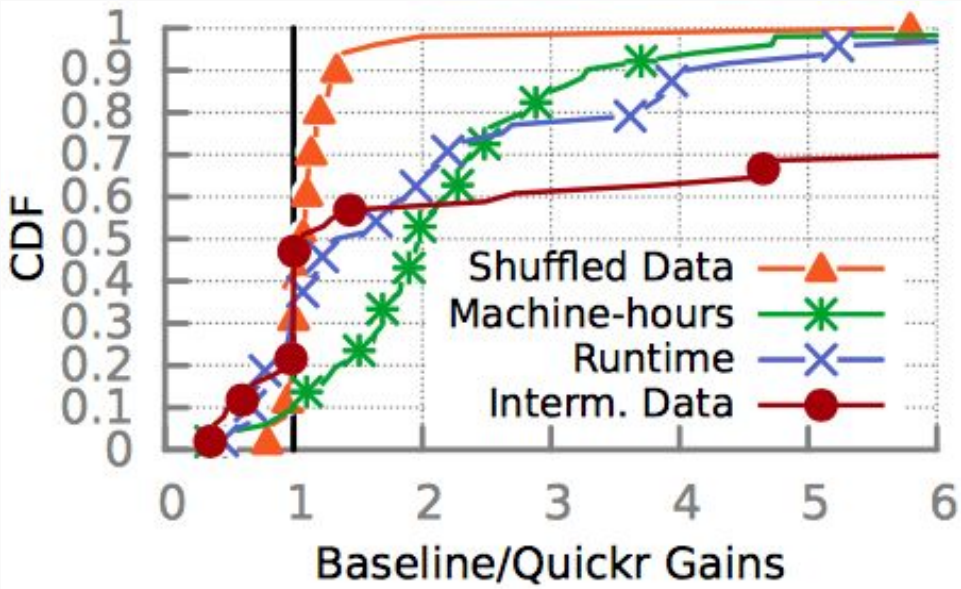
Error metrics

- Missed groups
  - Fraction of groups in answer that are missed
- Aggregation Error
  - Average error b/w estimated and true value of all aggregations in a query

Cluster

- Same as production
- “Datacenter-standard”
  - Ten cores
  - 100 GB memory
  - Two high RPM disks
  - Some SSDs
  - Two 10 Gbps NICs

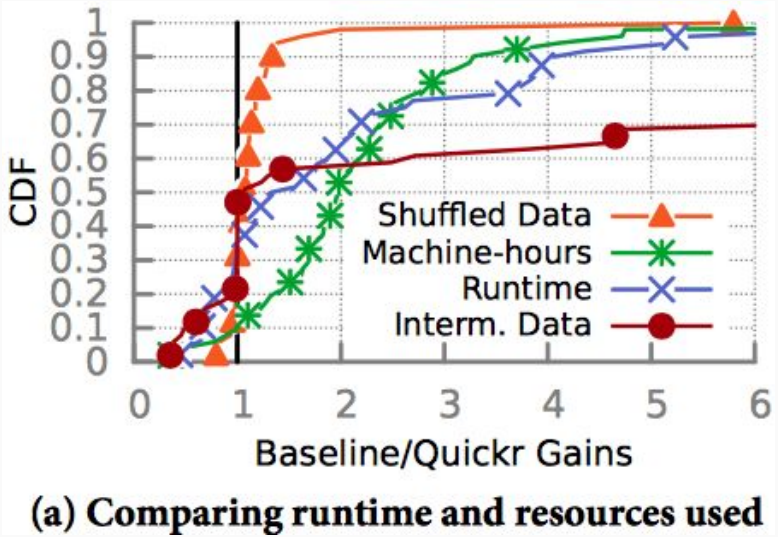
Performance gains



(a) Comparing runtime and resources used

Performance gains - Runtime

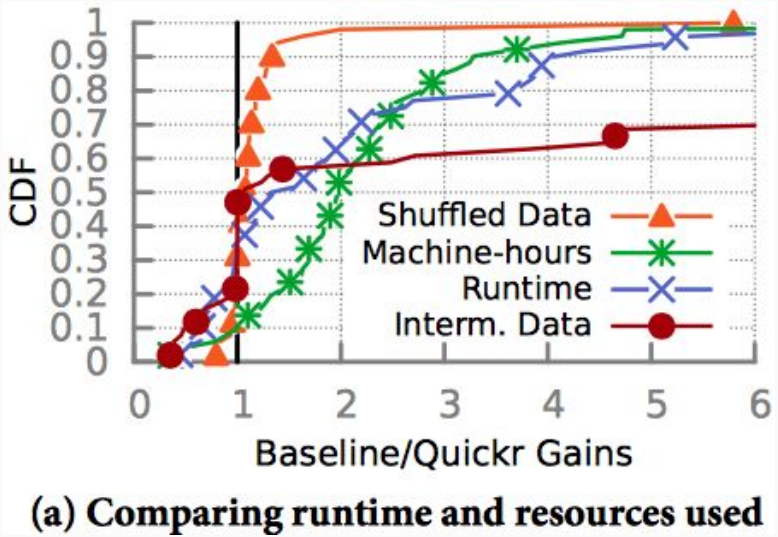
- Runtime improvements follow resource usage
  - Queries that process small amount of data are critical-path limited
- Median improvement of 1.6x
- 20% are worse
  - Runtime influenced by task failures and outliers
  - Fair-sharing schedulers -> resources vary



(a) Comparing runtime and resources used

Performance gains - Machine hours

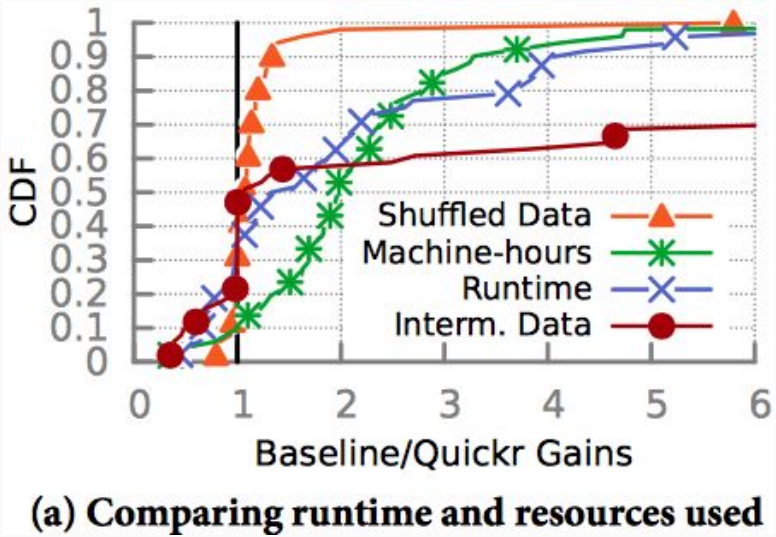
- Quickr lowers resource usage of median query by 2x
- Runtime is noisy metric, prefer machine hours
- More stable for efficiency comparisons



(a) Comparing runtime and resources used

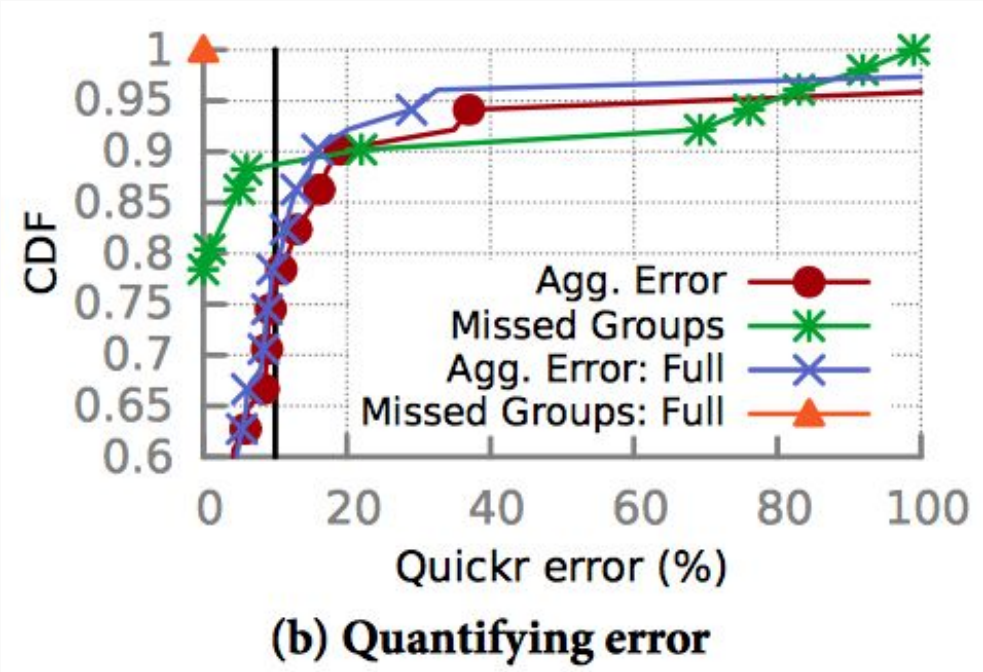
Performance gains - Intermediate/Shuffled

- Intermediate Data
  - Much less for sampled plans
  - 50% write more or same with Quickr
  - 40% decrease by 4x or more
- Shuffled Data
  - Less improvements than intermediate data
- Quickr decreases degree-of-parallelism
  - Hash joins replace pair joins
    - Faster completion time
    - More data to shuffle



(a) Comparing runtime and resources used

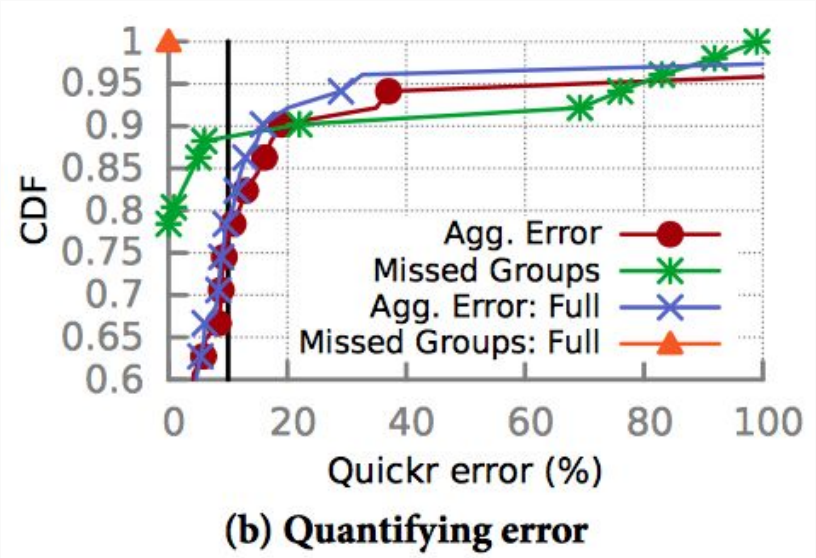
Quantifying error



(b) Quantifying error

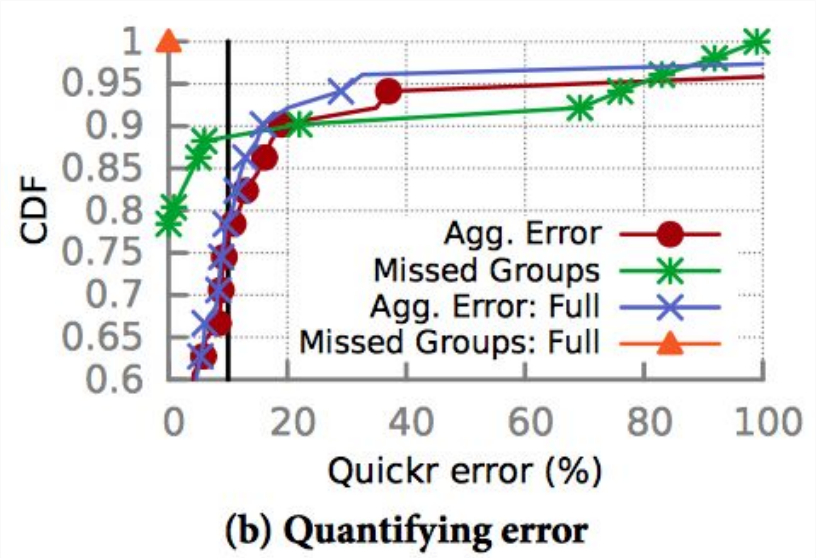
Quantifying error - Missed Groups

- 20% queries missing groups
- `LIMIT 100` after sorting aggregation column
- *Full* output answer before `LIMIT 100`
  - No groups missing



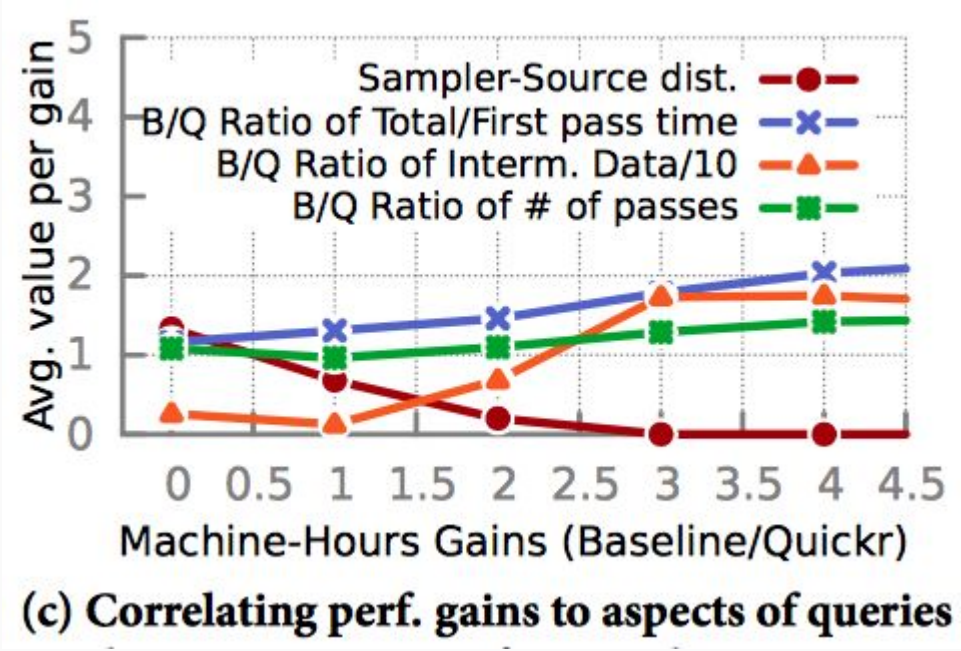
Quantifying error - Aggregate Error

- 80% queries within ±10%, 90% queries within ±20%,
- Support skewed across groups
- `SUM`s computed over values with high skew have high error



(b) Quantifying error

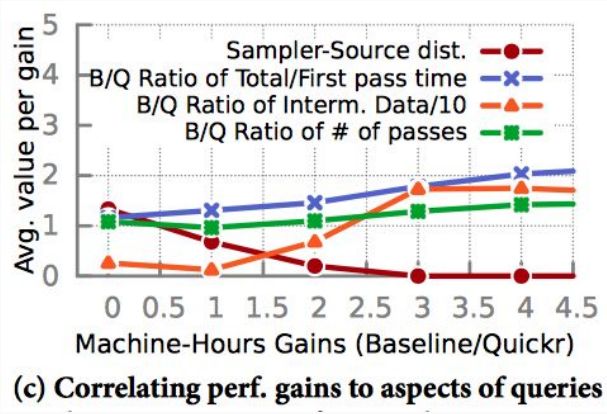
What does Quickr do?



(c) Correlating perf. gains to aspects of queries

What does Quickr do?

- Sampler-Source
  - Gains increase when samplers closer to source
- Total/First pass time, # of passes
  - Gains larger for deeper queries
- Intermediate Data/10
  - Highest gaining queres greatly reduce intermediate data (19x max)



(c) Correlating perf. gains to aspects of queries

What does Quickr do?

- Quickr considers samplers natively, so explores more alternatives
- “Sampler-Source distance” number of IO passes b/w extraction stage and sampler

Metric	Percentile value					
	10th	25th	50th	75th	90th	95th
Baseline QO time	0.38	0.49	0.51	0.54	0.56	0.57
Quickr QO time	0.48	0.5	0.52	0.55	0.57	0.58

Table 4: Query Optimization (QO) times (sec.)

Metric	Value					
	0	1	2	3	4	9
Samplers per query	25%	51%	9%	11%	2%	2%
Sampler-Source dist.	60%	12%	10%	17%	0%	0%

Table 5: Number of samplers per query and their locations

What samplers? How often?

- Uniform used 2x as much as distinct, universe
- Uniform replaces distinct b/c accurate enough with better performance
- Universe only used for queries that join two large relations

Metric	Sampler Type		
	Uniform	Distinct	Universe
Distribution across samplers	54%	26%	20%
Queries that use at least 1 sampler of a certain type	49%	24%	9%

Table 7: Frequency of use of various samplers.

Quickr vs Apriori samples

- Apriori samples: poor coverage for any feasible storage budget
- So compare against Blinkdb…
  - `store_sales` table of TPC-DS
  - Samples w/ best possible performance

Quickr vs Apriori samples

- Queries ran in Hive atop Tez
- Samples not explicitly stored in memory, file-system cache helps
- Median performance gain = 0%
- 24% improvement gain for 14 queries
  - At 10x storage budget!

Storage Budget	Coverage	Median Perf. gain: All	Median Perf. gain: Covered	Median Error
Default parameters (specifically, K=M=10^5).				
0.5×	0/64	0%	–	–
1×	0/64	0%	–	–
4×	9/64	0%	27%	6%
10×	14/64	0%	24%	5%
Tuned for small group size (K=M=10^1).				
0.5×	8/64	0%	35%	6%
1×	7/64	0%	35%	6%
4×	11/64	0%	32%	6%
10×	12/64	0%	24%	6%

Table 6: BlinkDB's performance on TPC-DS.

Quickr vs Apriori samples

- Predictable aggregation queries (touching only one dataset) greatly benefit from Apriori samples
  - Each stratified sample will be small and many different samples can be feasibly stored per dataset
- But this does not happen in TPC-DS or in MSFT production clusters
- Additional complication:
  - Datasets churn continuously

Related

Quickr vs Rest

- STRAT, SciBORQ, BlinkDB, OLA...
- Universe sampler operator
- Decide which queries can be sampled
- ASALQA
- `JOIN`s

Quickr vs BlinkDB

- Complex queries (i.e. `JOIN`)
  - Universe sampler
- Zero apriori overhead
- Lazy approximations

Quickr vs Online Aggregation (OLA)

- OLA progressively processes input, updating the answer
- Quickr views input once, maintains subset through query
- OLA requires specialized `JOIN`s and in-memory data-structures

Approximate Query Processing: No Silver Bullet

Surajit Chaudhuri, Bolin Ding, Srikanth Kandula
Microsoft Research

Lessons Learned

- Processing power can’t keep up with growth of data
- AQP has the potential to save time, money and be truly disruptive
- AQP systems are characterized by:
  - Generality of language they support
  - Error model, accuracy guarantee
  - Amount of work saved online and required offline
- Impossible to have a system with:
  - Flexible language, lots of work saved, and accuracy high enough for a production application
- “There are no silver bullets”

Main problem:

AQP systems have been hard to adopt

“Where to go from here?”

Theoretical considerations:

- Build primitives for developers to use in order to make application specific optimizations (this is *Quickr*)
- Provide accuracy guarantees in a query-independent way

Practical Considerations:

- Integrate AQP systems with existing platforms
-

“Where to go from here?”

Practical Suggestions

- Integrate AQP with existing platforms
- Approximate execution mode for querying
- Experiment with new scenarios

Thank you!

Andrew Levin, William Schmitt, Zhao Zhang