Query Optimization for Dynamic Imputation

José Cambronero[1] John K. Feser[1] Micah J. Smith[1]
Samuel Madden
{jcamsan,feser,madden}@csail.mit.edu
micahs@mit.edu

August 2017

[1]Author contributed equally

Missing Data: Common in Real World Databases

- Imagine a Center for Disease Control (CDC) analyst collecting data for study

Missing Data: Common in Real World Databases

- Imagine a Center for Disease Control (CDC) analyst collecting data for study
  - Database integration

nhanes
National Health and Nutrition Examination Survey

AMERICAN COMMUNITY SURVEY
U.S. CENSUS BUREAU

white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
?	?	?	...
630	17438	?	...
600	?	24	...
...	...	...	...

Missing Data: Common in Real World Databases

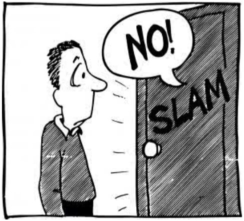
- Imagine a Center for Disease Control (CDC) analyst collecting data for study
  - Denormalized DBs

→

white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
?	?	?	...
630	17438	?	...
600	?	24	...
...	...	...	...

Missing Data: Common in Real World Databases

- Imagine a Center for Disease Control (CDC) analyst collecting data for study
  - Survey non-response



white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
?	?	?	...
630	17438	?	...
600	?	24	...
...	...	...	...

CDC Tables Missing Value Distributions

(a) Demographics (`demo`). 10175 rows.

Attribute	Missing
age_months	93.39 %
age_yrs	0.00 %
gender	0.00 %
id	0.00 %
income	1.31 %
is_citizen	0.04 %
marital_status	43.30 %
num_people_household	0.00 %
time_in_us	81.25 %
years_edu_children	72.45 %

(b) Laboratory Results (`labs`). 9813 rows.

Attribute	Missing
albumin	17.95 %
blood_lead	46.86 %
blood_selenium	46.86 %
cholesterol	22.31 %
creatine	72.59 %
hematocrit	12.93 %
id	0.00 %
triglyceride	67.94 %
vitamin_b12	45.83 %
white_blood_cell_ct	12.93 %

(c) Physical Results (`exams`). 9813 rows.

Attribute	Missing
arm_circumference	5.22 %
blood_pressure_secs	3.11 %
blood_pressure_systolic	26.91 %
body_mass_index	7.72 %
cuff_size	23.14 %
head_circumference	97.67 %
height	7.60 %
id	0.00 %
waist_circumference	11.74 %
weight	0.92 %

Diseases Stop for No Missing Value

- CDC analyst still needs to execute their queries

```
SELECT
  income,
  AVG(white_blood_cell_ct)
FROM demo, exams, labs
WHERE
  gender = 2 AND
  weight >= 120 AND
  demo.id = exams.id AND
  exams.id = labs.id
GROUP BY demo.income
```

white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
?	?	?	...
630	17438	?	...
600	?	24	...
...	...	...	...

Stuck Between a Rock and a Hard Place: Simple

- Change query to remove tuples with missing values in relevant fields

Stuck Between a Rock and a Hard Place: Simple

- Change query to remove tuples with missing values in relevant fields
- Lost almost 15% of tuples

```
SELECT
  income,
  AVG(white_blood_cell_ct)
FROM demo, exams, labs
WHERE
  gender = 2 AND
  weight >= 120 AND
  demo.id = exams.id AND
  exams.id = labs.id AND
  // extra predicates to filter nulls
  income IS NOT NULL AND
  white_blood_cell_ct IS NOT NULL
GROUP BY demo.income
```

Stuck Between a Rock and a Hard Place: Simple

- Change query to remove tuples with missing values in relevant fields
- Lost almost 15% of tuples
- If not missing uniformly at random, AVG can produce skewed estimate

```
SELECT
  income,
  AVG(white_blood_cell_ct)
FROM demo, exams, labs
WHERE
  gender = 2 AND
  weight >= 120 AND
  demo.id = exams.id AND
  exams.id = labs.id AND
  // extra predicates to filter nulls
  income IS NOT NULL AND
  white_blood_cell_ct IS NOT NULL
GROUP BY demo.income
```

Stuck Between a Rock and a Hard Place: Simple

In general:

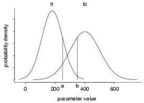
- `SELECT * FROM data_dirty WHERE critical IS NOT NULL`
- ✓ Fast and simple strategy
- ✗ Lose (hard earned) data in non-null covariates dropped
- ✗ Biased results

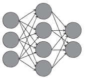
Stuck Between a Rock and a Hard Place: Complex

- Imputation: fill in missing values with model, alternative to drop

Stuck Between a Rock and a Hard Place: Complex

white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
?	?	?	...
630	17438	?	...
600	?	24	...
...	...	...	...





white_blood_cell_ct	blood_selenium	triglyceride	...
520	19562	64	...
#	#	#	...
630	17438	#	...
600	#	24	...
...	...	...	...

Stuck Between a Rock and a Hard Place: Complex

In general:

- `data_cleaned <- fancy_statistical_model(data_dirty)`
- ✓ Robust and can address domain specific requirements
- ✗ Slow (development and training/application time) and complex
- Amortization doesn't help: can't pay cost once when unfamiliar with the data and exploring multiple models

Best of Both Worlds

- Explore data quickly without committing to strategy

Best of Both Worlds

- Explore data quickly without committing to strategy
- Dynamically pick speed/simplicity versus quality/complexity

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization
  - impute only what is needed

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization
  - impute only what is needed
  - choose the plans that best balance tradeoffs along speed and quality

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization
  - impute only what is needed
  - choose the plans that best balance tradeoffs along speed and quality
- Implemented as part of experimental DB used at MIT

ImputeDB

ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization
  - impute only what is needed
  - choose the plans that best balance tradeoffs along speed and quality
- Implemented as part of experimental DB used at MIT
- 10x - 140x speedup w.r.t. standard practice

ImputeDB

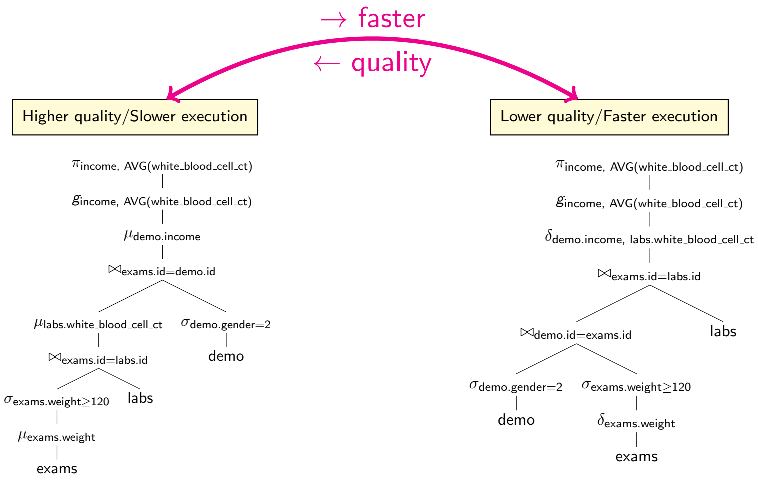
ImputeDB: enable data exploration by optimizing placement of imputation and dropping operations

Key insight: only need to impute data relevant to query

- Query plan optimization
  - impute only what is needed
  - choose the plans that best balance tradeoffs along speed and quality
- Implemented as part of experimental DB used at MIT
- 10x - 140x speedup w.r.t. standard practice
- 0% - 8% error w.r.t. standard practice

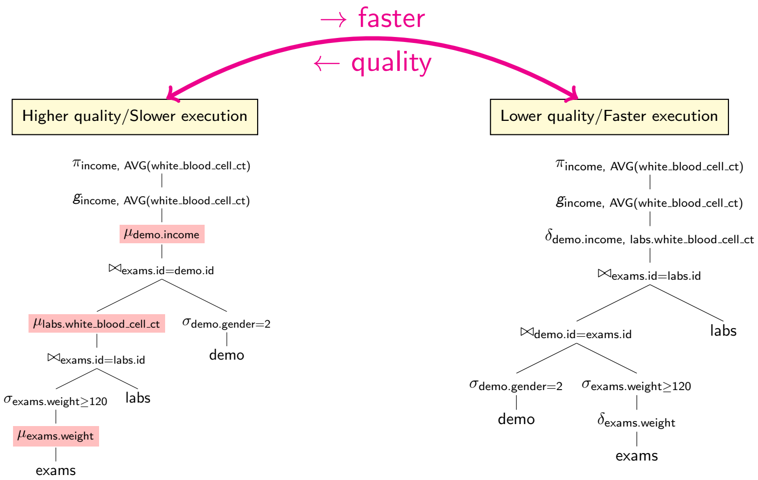
Express Quality and Performance Trade-off

- Planning extended to incorporate: *Impute* (μ_C) and *Drop* (δ_C)
- New operators output tuples without missing values for cols $\in C$



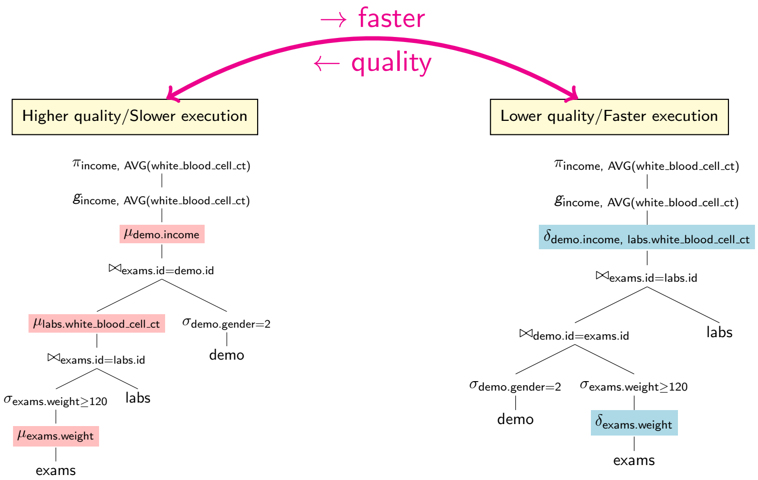
Express Quality and Performance Trade-off

- Planning extended to incorporate: *Impute* (μ_C) and *Drop* (δ_C)
- New operators output tuples without missing values for cols $\in C$



Express Quality and Performance Trade-off

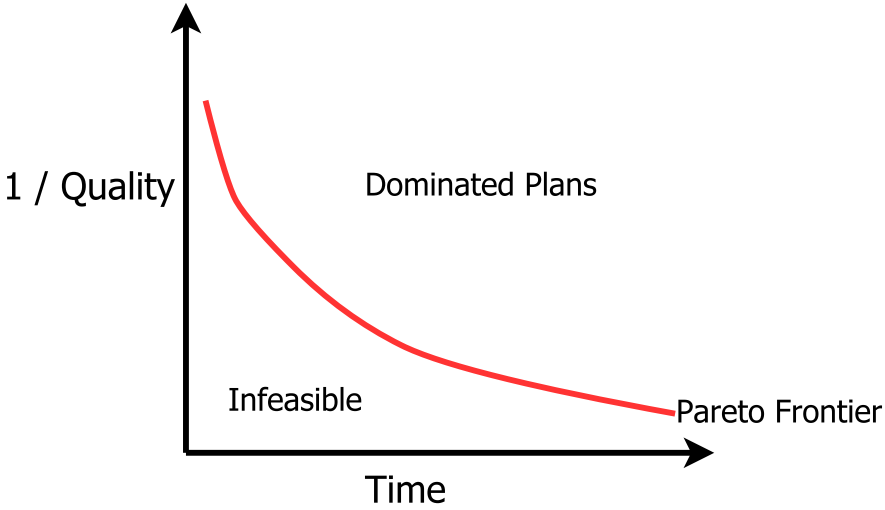
- Planning extended to incorporate: *Impute* (μ_C) and *Drop* (δ_C)
- New operators output tuples without missing values for cols $\in C$



Operation Placement Trade-offs

Op.	Early placement	Late placement
Impute	$\uparrow$ data, $\downarrow$ speed	$\downarrow$ data, $\uparrow$ speed, $\uparrow$ potential relevance of data
Drop	$\downarrow$ cardinality for later stages, may lose more than necessary	$\downarrow$ speed, $\downarrow$ early drops of potentially relevant data

Optimization Problem



Optimization Problem

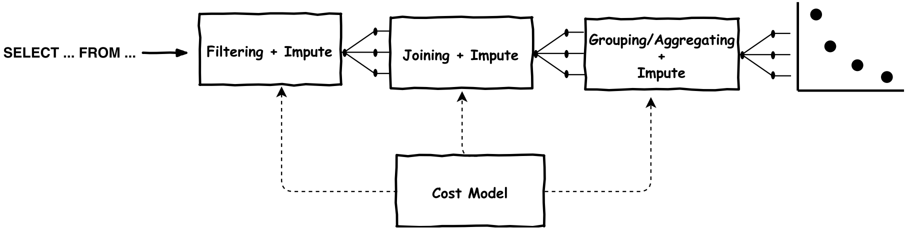
Definition

Produce a Pareto frontier over key dimensions of imputation quality and execution time for query plans subject to:

- No standard relational operator sees missing values
- Attributes imputed restricted to local requirements or all missing needed later in plan

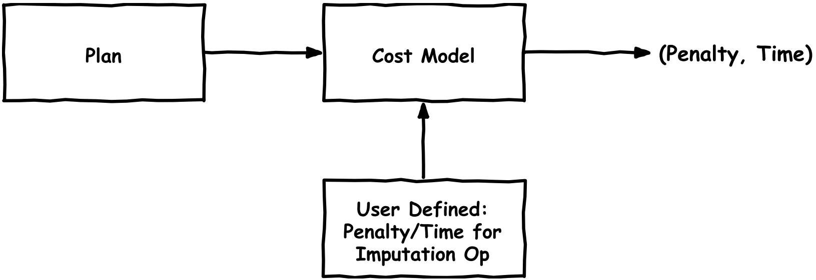
Optimizer Pipeline

- Cost-based, builds on seminal Selinger join optimization
- Each stage maintains collection of Pareto frontiers



Characterizing Imputation Operations

- Plan q cost: $\langle \textsc{Penalty}(q), \textsc{Time}(q) \rangle$



Characterizing Imputation Operations

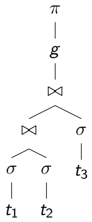
- Plan q cost: $\langle \textsc{Penalty}(q), \textsc{Time}(q) \rangle$

Cost	Higher	Lower
Imputation Penalty	More columns imputed	More data
Time Estimate	More tuples/attributes, complex model	Less data, simple model

Optimizer Search Space

- Filters pushed down
- Left-deep joins
- Aggregations/projections last
- *Impute*(μ) and *Drop*(δ) placed before standard operations

Example plan:

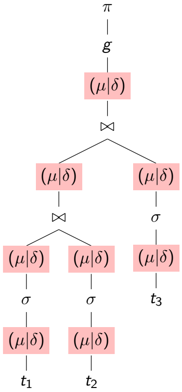


[1] Join order can change based on operation optimization

Optimizer Search Space

- Filters pushed down
- Left-deep joins
- Aggregations/projections last
- *Impute*(μ) and *Drop*(δ) placed before standard operations

Example plan[1]:



[1] Join order can change based on operation optimization

Optimizer Complexity

- Still exponential, $\mathcal{O}(2^{4J})$ where J is number of joins
- In practice not an issue for up to 5 joins (sub-second planning)
- Mean planning time for experiments up to 2.8ms

Joint Optimization: An Illustrative Example

```
select avg(weight)
  from exams, demo
where
  demo.years_edu > 10 and
  exams.height >= 155.88 and
  exams.id = demo.id
```

Joint Optimization: An Illustrative Example

- Determine imputation needs:

  `exams.weight, exams.height, demo.years_edu`

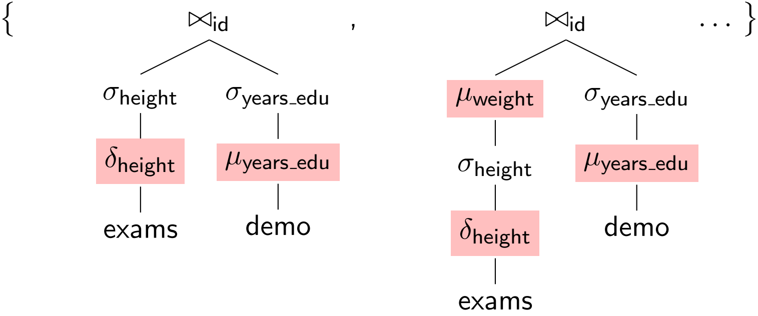
Joint Optimization: An Illustrative Example

- Push down filters and generate possible plans for selections

$$\left\{ \begin{array}{c} \sigma_{\text{height}} \\ | \\ \boxed{\delta_{\text{height}}} \\ | \\ \text{exams} \end{array} , \quad \begin{array}{c} \sigma_{\text{height}} \\ | \\ \boxed{\mu_{\text{height, weight}}} \\ | \\ \text{exams} \end{array} , \quad \ldots \right\}$$

Joint Optimization: An Illustrative Example

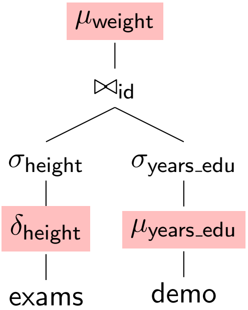
- Jointly optimize join order and add imputation operators



Joint Optimization: An Illustrative Example

- Pick set of plans that have all requisite tables and form current frontier
- Add imputations necessary to aggregate and prune

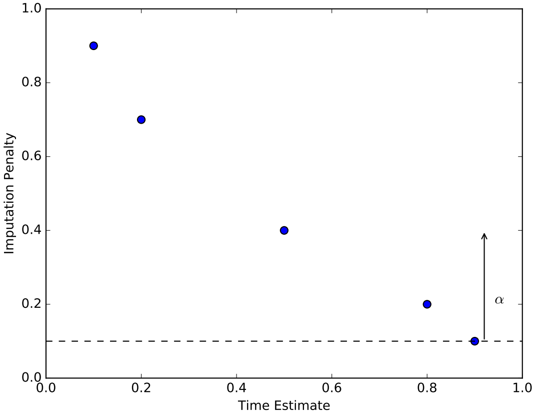
$\{ \ldots ,$



$, \ldots \}$

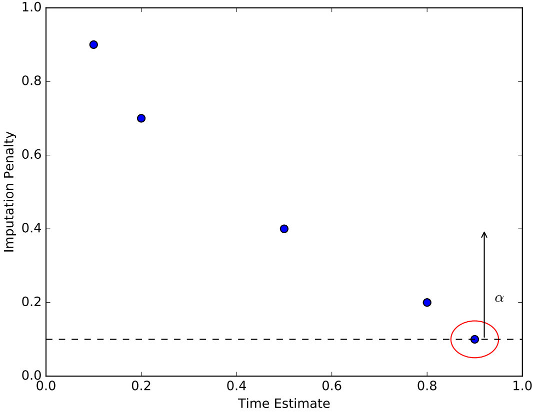
Joint Optimization: An Illustrative Example

- Extract final frontier



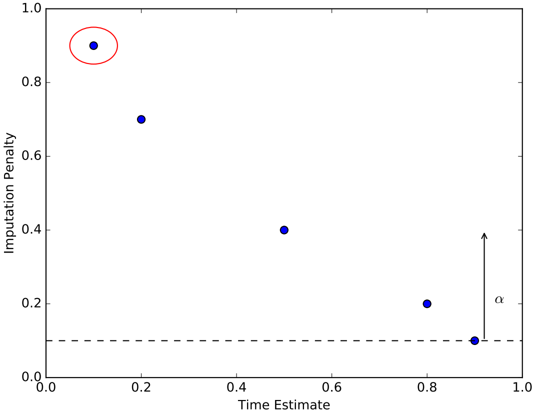
Joint Optimization: An Illustrative Example

- Extract final frontier
- $\alpha = 0.0$: quality optimized

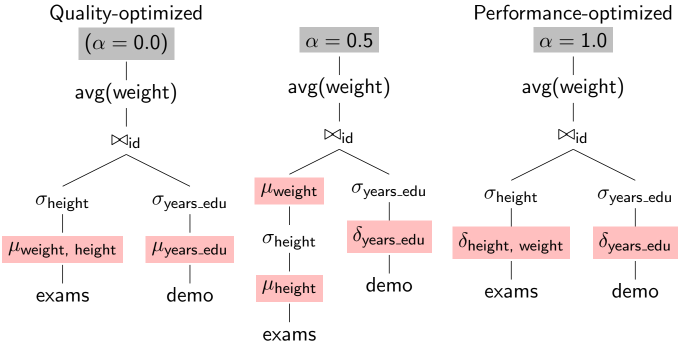


Joint Optimization: An Illustrative Example

- Extract final frontier
- $\alpha = 0.0$: quality optimized
- $\alpha = 1.0$: performance optimized



Joint Optimization: An Illustrative Example



Experimental Results

- Imputation models: chained-equation decision trees, non-blocking approximate mean, hot deck
- Varying amounts of missing data
- Queries with interesting joins and aggregations
- Evaluate `COUNT`, `AVG`
  - `MAX`, `MIN` not expected to perform well
  - Extremal value imputation outside the scope

Errors: 0-8% for quality-optimized relative to base

	lower better			closer to 1.0 better		
	$\underbrace{\hspace{3em}}_{\text{SMAPE}[1]}$			$\underbrace{\hspace{3em}}_{\text{Count Fraction}[2]}$		
Query	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1	0.39	1.19	1.24	1.00	0.89	0.88
2	1.49	1.65	2.41	1.00	0.99	0.33
3	1.05	1.05	2.14	1.00	1.00	0.57
4	0.22	0.30	0.30	0.96	0.89	0.80
5	0.02	0.19	0.20	1.01	0.94	0.94
6	1.04	0.88	4.43	0.79	0.80	0.64
7	0.03	0.43	0.40	1.00	0.66	0.65
8	7.97	7.97	19.99	1.00	1.00	0.22
9	2.03	2.17	2.49	1.00	0.85	0.84

[1] Symmetric Mean Absolute Percentage Error: symmetric for over/under imputation

[2] Fraction of tuples used to compute aggregate relative to base table strategy

Errors: 0-8% for quality-optimized relative to base

	SMAPE[1]			Count Fraction[2]		
Query	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1	0.39	1.19	1.24	1.00	0.89	0.88
2	1.49	1.65	2.41	1.00	0.99	0.33
3	1.05	1.05	2.14	1.00	1.00	0.57
4	0.22	0.30	0.30	0.96	0.89	0.80
5	0.02	0.19	0.20	1.01	0.94	0.94
6	1.04	0.88	4.43	0.79	0.80	0.64
7	0.03	0.43	0.40	1.00	0.66	0.65
8	7.97	7.97	19.99	1.00	1.00	0.22
9	2.03	2.17	2.49	1.00	0.85	0.84

- Quality optimized: low errors correlate with α

[1] Symmetric Mean Absolute Percentage Error: symmetric for over/under imputation

[2] Fraction of tuples used to compute aggregate relative to base table strategy

Errors: 0-8% for quality-optimized relative to base

	SMAPE[1]			Count Fraction[2]		
Query	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1	0.39	1.19	1.24	1.00	0.89	0.88
2	1.49	1.65	2.41	1.00	0.99	0.33
3	1.05	1.05	2.14	1.00	1.00	0.57
4	0.22	0.30	0.30	0.96	0.89	0.80
5	0.02	0.19	0.20	1.01	0.94	0.94
6	1.04	0.88	4.43	0.79	0.80	0.64
7	0.03	0.43	0.40	1.00	0.66	0.65
8	7.97	7.97	19.99	1.00	1.00	0.22
9	2.03	2.17	2.49	1.00	0.85	0.84

- Quality optimized: low errors correlate with α
- Cost model makes small errors occasionally

[1] Symmetric Mean Absolute Percentage Error: symmetric for over/under imputation

[2] Fraction of tuples used to compute aggregate relative to base table strategy

Errors: 0-8% for quality-optimized relative to base

	SMAPE[1]			Count Fraction[2]		
Query	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1	0.39	1.19	1.24	1.00	0.89	0.88
2	1.49	1.65	2.41	1.00	0.99	0.33
3	1.05	1.05	2.14	1.00	1.00	0.57
4	0.22	0.30	0.30	0.96	0.89	0.80
5	0.02	0.19	0.20	1.01	0.94	0.94
6	1.04	0.88	4.43	0.79	0.80	0.64
7	0.03	0.43	0.40	1.00	0.66	0.65
8	7.97	7.97	19.99	1.00	1.00	0.22
9	2.03	2.17	2.49	1.00	0.85	0.84

- Quality optimized: low errors correlate with α
- Cost model makes small errors occasionally
- Skewed tuple distribution + performance optimized: can drop a lot of relevant tuples

[1] Symmetric Mean Absolute Percentage Error: symmetric for over/under imputation

[2] Fraction of tuples used to compute aggregate relative to base table strategy

Errors: 0-8% for quality-optimized relative to base

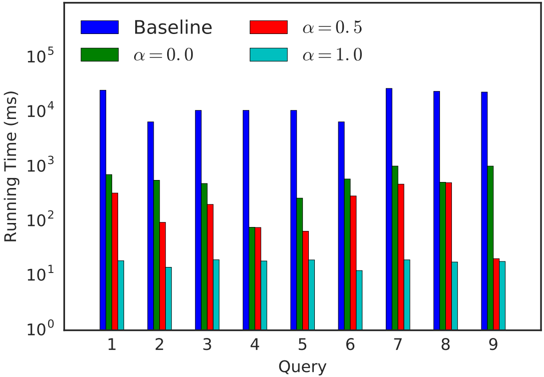
	SMAPE[1]			Count Fraction[2]		
Query	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$
1	0.39	1.19	1.24	1.00	0.89	0.88
2	1.49	1.65	2.41	1.00	0.99	0.33
3	1.05	1.05	2.14	1.00	1.00	0.57
4	0.22	0.30	0.30	0.96	0.89	0.80
5	0.02	0.19	0.20	1.01	0.94	0.94
6	1.04	0.88	4.43	0.79	0.80	0.64
7	0.03	0.43	0.40	1.00	0.66	0.65
8	7.97	7.97	19.99	1.00	1.00	0.22
9	2.03	2.17	2.49	1.00	0.85	0.84

- Quality optimized: low errors correlate with α
- Cost model makes small errors occasionally
- Skewed tuple distribution + performance optimized: can drop a lot of relevant tuples
- Plan selected depends on distribution of frontier

[1] Symmetric Mean Absolute Percentage Error: symmetric for over/under imputation

[2] Fraction of tuples used to compute aggregate relative to base table strategy

Running Times: 10-140x faster for quality optimized



- Quality optimized: 75ms-1s (versus 6.5s-27s)
- Performance optimized: 12-19ms

ImputeDB: Takeaways

- Extended relational algebra for imputation
- Joint optimization of standard operations and impute results in order-of-magnitude speedups
- Plans produced express underlying tradeoffs in quality and execution

Future Work Directions

- Imputation confidence through resampling
- Imputation operators: expand beyond *Drop* and *Impute*
- Adaptive query planning: shared across queries, beyond caching

Acknowledgements

- Thanks to Akande, Li, and Reiter for giving us a cleaned copy of the 2012 ACS PUMS
- This work was supported in part by the Qatar Research Computing Institute and the Center for Resilient Software at MIT