Vertical Partitioning for Query Processing over Raw Data

Weijie Zhao, Yu Cheng, and Florin Rusu

University of California, Merced

June 30, 2015

Outline

Example SDSS Schema

- Sloan Digital Sky Survey (SDSS)
- PhotoPrimary table: 509 attributes!

name	type	length	unit	ucd	description
objID	bigint	8			Unique SDSS identifier composed from [skyVersion,rerun,run,camcol,field,obj].
skyVersion	tinyint	1			Layer of catalog (currently only one layer, 0; 0-15 available)
run	smallint	2			Run number
rerun	smallint	2			Rerun number
camcol	tinyint	1			Camera column
field	smallint	2			Field number
obj	smallint	2			The object id within a field. Usually changes between reruns of the same field
mode	tinyint	1			1: primary, 2: secondary, 3: other
nChild	smallint	2			Number of children if this is a composite object that has been deblended. BRIGHT (in a flags sense) objects also have nchild == 1, the non-BRIGHT sibling.
type	smallint	2			Type classification of the object (star, galaxy, cosmic ray, etc.)
clean	int	4			Clean photometry flag (1=clean, 0=unclean).
probPSF	real	4			Probability that the object is a star. Currently 0 if type == 3 (galaxy), 1 if type == 6 (star).
insideMask	tinyint	1			Flag to indicate whether object is inside a mask and why
flags	bigint	8			Photo Object Attribute Flags
rowc	real	4	pix		Row center position (r-band coordinates)
rowcErr	real	4	pix		Row center position error (r-band coordinates)
colc	real	4	pix		Column center position (r-band coordinates)
colcErr	real	4	pix		Column center position error (r-band coordinates)
rowv	real	4	deg/day		Row-component of object's velocity
rowvErr	real	4	deg/day		Row-component of object's velocity error
colv	real	4	deg/day		Column-component of object's velocity
colvErr	real	4	deg/day		Column-component of object's velocity error
rowc_u	real	4	pix		Row center, u-band
rowc_g	real	4	pix		Row center, g-band
rowc_r	real	4	pix		Row center, r-band
rowc_i	real	4	pix		Row center, i-band

Example SDSS Data in CSV Format

UCMERCED

```
objID,skyVersion,run,rerun,camcol,field,obj,mode,nChild,type,clean,probPSF,insideMask,fl
1237645942366274027,2,109,301,3,114,491,1,0,3,1,0,0,217164284160,226.89418,0.13062,1969.
1237645942366274688,2,109,301,3,114,1152,1,0,3,0,0,0,281543964623104,145.553284,0.520364
1237645942366274694,2,109,301,3,114,1158,1,0,6,0,1,0,281543964623616,225.317307,0.507917
1237645942370140312,2,109,301,3,173,152,1,0,6,1,1,0,158398662443776,720.658875,4.751435E
1237645942370140314,2,109,301,3,173,154,1,0,6,1,1,0,68988047872,734.537415,4.991623E-3,1
1237645942370140321,2,109,301,3,173,161,1,0,6,1,1,0,193585182019600,771.662109,3.504038E
1237645942370140322,2,109,301,3,173,162,1,0,3,0,0,0,144572653934612752,763.089355,0.5900
1237645942370140323,2,109,301,3,173,163,1,0,3,0,0,0,144431916446257424,797.722046,0.5742
1237645942370140324,2,109,301,3,173,164,1,0,3,0,0,0,316728370401624,759.693542,0.430969,
1237645942370140325,2,109,301,3,173,165,1,0,3,0,0,0,144431916446257424,772.056763,0.7608
1237645942370140327,2,109,301,3,173,167,1,0,6,1,1,0,68987912704,813.796387,0.016672,693.
1237645942370140330,2,109,301,3,173,170,1,0,6,1,1,0,35253360136208,845.047607,5.510448E-
1237645942370140331,2,109,301,3,173,171,1,0,3,0,0,0,105622104310104,840.993713,0.351185,
1237645942370140332,2,109,301,3,173,172,1,0,6,0,1,0,387097114444048,841.738831,0.33631,1
1237645942370140503,2,109,301,3,173,343,1,0,6,1,1,0,68987912960,975.684814,0.068882,665.
1237645942370140504,2,109,301,3,173,344,1,0,3,1,0,0,217164284160,984.625671,0.367924,97.
1237645942370140505,2,109,301,3,173,345,1,0,6,1,1,0,68987912448,1008.3241,0.055251,295.5
1237645942370140507,2,109,301,3,173,347,1,0,3,1,0,0,68987912448,1098.70068,0.191982,131.
1237645942370140509,2,109,301,3,173,349,1,0,6,1,1,0,72092847397929232,1136.47461,0.04868
1237645942370140510,2,109,301,3,173,350,1,0,3,1,0,0,35184640524816,1147.8905,0.054839,36
1237645942370140512,2,109,301,3,173,352,1,0,3,1,0,2,68988043520,1212.91077,0.191284,584.
1237645942370140514,2,109,301,3,173,354,1,0,6,1,1,0,68987912192,1220.56543,0.065685,318.
1237645942370140516,2,109,301,3,173,356,1,0,3,1,0,0,2278815830856,1231.65027,0.151327,78
1237645942370140518,2,109,301,3,173,358,1,0,3,1,0,0,35255507620624,1255.73218,0.109877,3
1237645942370140519,2,109,301,3,173,359,1,0,6,0,1,0,545426755424528,1270.13306,0.565565,
```

Example SQL Query on SDSS

```
SELECT TOP 10 P.ObjID
FROM PhotoPrimary AS P JOIN Neighbors AS N ON P.ObjID = N.ObjID
    JOIN PhotoPrimary AS L ON L.ObjID = N.NeighborObjID
WHERE P.ObjID < L. ObjID AND
    abs((P.u-P.g)-(L.u-L.g))<0.05 AND
    abs((P.g-P.r)-(L.g-L.r))<0.05 AND
    abs((P.r-P.i)-(L.r-L.i))<0.05 AND
    abs((P.i-P.z)-(L.i-L.z))<0.05
```

Example SQL Query on SDSS

```
SELECT TOP 10 P.ObjID
FROM PhotoPrimary AS P JOIN Neighbors AS N ON P.ObjID = N.ObjID
    JOIN PhotoPrimary AS L ON L.ObjID = N.NeighborObjID
WHERE P.ObjID < L. ObjID AND
    abs((P.u-P.g)-(L.u-L.g))<0.05 AND
    abs((P.g-P.r)-(L.g-L.r))<0.05 AND
    abs((P.r-P.i)-(L.r-L.i))<0.05 AND
    abs((P.i-P.z)-(L.i-L.z))<0.05
```

- Workload of 1 million queries uses only 74 out of 509 attributes!

Outline

1. SDSS Data and Queries
2. Problem Statement
3. MIP Formulation
4. Heuristic Algorithm
5. Pipeline Processing
6. Experiments
7. Conclusions

Problem Statement

UCMERCED

Raw data processing with partial loading

Given a dataset in some raw format, a query workload, and a limited database storage budget, find what data to load in the database such that the overall workload execution time is minimized.

Problem Statement

Raw data processing with partial loading

Given a dataset in some raw format, a query workload, and a limited database storage budget, find what data to load in the database such that the overall workload execution time is minimized.

- Accessing data from the database is clearly optimal in the case of workloads with tens of queries.

Problem Statement

UCMERCED

Raw data processing with partial loading

Given a dataset in some raw format, a query workload, and a limited database storage budget, find what data to load in the database such that the overall workload execution time is minimized.

- Accessing data from the database is clearly optimal in the case of workloads with tens of queries.
- Datasets are extremely large nowadays. Full data replication requires significant amount of storage and takes a prohibitively long time.

Problem Statement

Raw data processing with partial loading

Given a dataset in some raw format, a query workload, and a limited database storage budget, find what data to load in the database such that the overall workload execution time is minimized.

- Accessing data from the database is clearly optimal in the case of workloads with tens of queries.
- Datasets are extremely large nowadays. Full data replication requires significant amount of storage and takes a prohibitively long time.
- Only a small portion of attributes are heavily used in most queries.

Related Work

- Raw data processing
  - External tables (MySQL, Oracle)
  - Cached in memory on a query-by-query basis (NoDB, DataVaults, SDS/Q, RAW, Impala)
  - Loading (adaptive partial loading, invisible loading, SCANRAW)

Related Work

- Raw data processing
  - External tables (MySQL, Oracle)
  - Cached in memory on a query-by-query basis (NoDB, DataVaults, SDS/Q, RAW, Impala)
  - Loading (adaptive partial loading, invisible loading, SCANRAW)
- Vertical partitioning
  - Top-down transaction-level algorithm (Chu et al.)
  - Top-down heuristics (Agrawal et al., Navathe et al.)
  - Bottom-up algorithms (Grund et al., Hammer et al., Hankins et al., Jindal et al., Papadomanolakis et al.)

- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.

Contributions

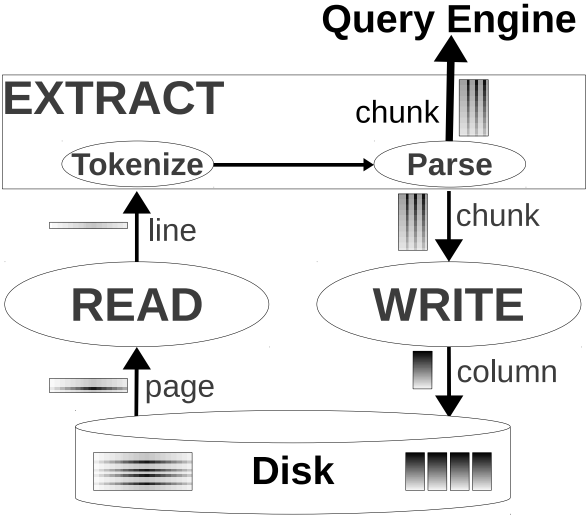
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency. The heuristic comes within close range of the optimal solution in a fraction of the time.

Contributions

- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency. The heuristic comes within close range of the optimal solution in a fraction of the time.
- We extend the optimization formulation and the heuristic to a restricted type of pipelined raw data processing.

Contributions

- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency. The heuristic comes within close range of the optimal solution in a fraction of the time.
- We extend the optimization formulation and the heuristic to a restricted type of pipelined raw data processing.
- We evaluate the performance of the heuristic and the accuracy of the optimization formulation over three real data formats: CSV, FITS, and JSON.


Query Engine
EXTRACT
chunk
Tokenize
Parse
line
chunk
READ
WRITE
page
column
Disk

Workload

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

Suppose $B = 3$, i.e., we can load 3 attributes.
What columns to choose?

Outline

Mixed Integer Programming Formulation: Variables

Variable	Description
raw_i; $i = \overline{0, m}$	read raw file at query i
t_{ij}; $i = \overline{0, m}$, $j = \overline{1, n}$	tokenize attribute j at query i
p_{ij}; $i = \overline{0, m}$, $j = \overline{1, n}$	parse attribute j at query i
$read_{ij}$; $i = \overline{1, m}$, $j = \overline{1, n}$	read attribute j at query i from processing format
$save_j$; $j = \overline{1, n}$	load attribute j in processing format

Mixed Integer Programming Formulation: Parameters

UCMERCED

Parameter	Description
$\|R\|$	number of tuples in relation R
S_{RAW}	size of raw file
SPF_j, $j = \overline{1, n}$	size of attribute j in processing format
B	size of storage in processing format
$band_{IO}$	storage bandwidth
T_{t_j}, $j = \overline{1, n}$	time to tokenize an instance of attribute j
T_{p_j}, $j = \overline{1, n}$	time to parse an instance of attribute j
w_i, $i = \overline{1, m}$	weight for query i

MIP Formulation

$$\text{minimize} \quad T_{load} + \sum_{i=1}^{m} w_i \cdot T_i \quad \text{subject to constraints:}$$

$$C_1 : \sum_{j=1}^{n} save_j \cdot SPF_j \cdot |R| \leq B$$

$$C_2 : \; read_{ij} \leq save_j; \; i = \overline{1, m}, \; j = \overline{1, n}$$

$$C_3 : \; save_j \leq p_{0j} \leq t_{0j} \leq raw_0; \; j = \overline{1, n}$$

$$C_4 : \; p_{ij} \leq t_{ij} \leq raw_i; \; i = \overline{1, m}, \; j = \overline{1, n}$$

$$C_5 : \; t_{ij} \leq t_{ik}; \; i = \overline{0, m}, \; j > k = \overline{1, n-1}$$

$$C_6 : \; read_{ij} + p_{ij} = 1; \; i = \overline{1, m}, \; j = \overline{1, n}, \; A_j \in Q_i$$

Objective Function

$$
\begin{aligned}
T_{load} = & raw_0 \cdot \frac{S_{RAW}}{band_{IO}} + \\
& |R| \cdot \sum_{j=1}^{n} \left(t_{0j} \cdot T_{t_j} + p_{0j} \cdot T_{p_j} + save_j \cdot \frac{SPF_j}{band_{IO}} \right) \\
T_i = & raw_i \cdot \frac{S_{RAW}}{band_{IO}} + \\
& |R| \cdot \sum_{j=1}^{n} \left(t_{ij} \cdot T_{t_j} + p_{ij} \cdot T_{p_j} + read_{ij} \cdot \frac{SPF_j}{band_{IO}} \right)
\end{aligned}
$$

UCMERCED

Definition (k-element cover)

Given a set of n elements $R = \{A_1, \ldots, A_n\}$, m subsets $W = \{Q_1, \ldots, Q_m\}$ of R, such that $\bigcup_{i=1}^{m} Q_i = R$, and a value k, the objective in the k-element cover problem is to find a size k subset R' of R that covers the largest number of subsets Q_i, i.e., $Q_i \subseteq R'$, $1 \leq i \leq m$.

Computational Complexity

Definition (k-element cover)

Given a set of n elements $R = \{A_1, \ldots, A_n\}$, m subsets $W = \{Q_1, \ldots, Q_m\}$ of R, such that $\bigcup_{i=1}^{m} Q_i = R$, and a value k, the objective in the k-element cover problem is to find a size k subset R' of R that covers the largest number of subsets Q_i, i.e., $Q_i \subseteq R'$, $1 \leq i \leq m$.

Definition (minimum k-set coverage)

Given a set of n elements $R = \{A_1, \ldots, A_n\}$, m subsets $W = \{Q_1, \ldots, Q_m\}$ of R, such that $\bigcup_{i=1}^{m} Q_i = R$, and a value k, the objective in the minimum k-set coverage problem is to choose k sets $\{Q_{i_1}, \ldots Q_{i_k}\}$ from W whose union has the smallest cardinality, i.e., $\left|\bigcup_{j=1}^{k} Q_{i_j}\right|$.

UCMERCED

Algorithm 1 Reduce k-element cover to minimum k'-set coverage

Input: Set $R = \{A_1, \ldots, A_n\}$ and m subsets $W = \{Q_1, \ldots, Q_m\}$ of R; number k' of sets Q_i to choose in minimum set coverage

Output: Minimum number k of elements from R covered by choosing k' subsets from W

```
for i = 1 to n do
    res = k-element cover(W, i)
    if res ≥ k' then return i
end for
```

Algorithm 2 Reduce k-element cover to minimum k'-set coverage

Input: Set $R = \{A_1, \ldots, A_n\}$ and m subsets $W = \{Q_1, \ldots, Q_m\}$ of R; number k' of sets Q_i to choose in minimum set coverage

Output: Minimum number k of elements from R covered by choosing k' subsets from W

```
for i = 1 to n do
    res = k-element cover(W, i)
    if res ≥ k' then return i
end for
```

- The MIP formulation is NP-hard and cannot be approximated unless NP-complete problems can be solved in randomized sub-exponential time.

Outline

Heuristic Algorithm: Query Coverage

Input: Workload $W = \{Q_1, \ldots, Q_m\}$; storage budget B
Output: Set of attributes $\{A_{j_1}, \ldots, A_{j_k}\}$ to be loaded in processing representation

1: $attsL = \emptyset$; $coveredQ = \emptyset$
2: **while** $\sum_{j \in attsL} SPF_j < B$ **do**
3: $\quad idx = \text{argmax}_{i \notin coveredQ} \left\{ \frac{cost(attsL) - cost(attsL \cup Q_i)}{\sum_{j \in \{attsL \cup Q_i \setminus attsL\}} SPF_j} \right\}$
4: $\quad$ **if** $cost\,(attsL) - cost\,(attsL \cup Q_{idx}) \leq 0$ **then break**
5: $\quad coveredQ = coveredQ \cup idx$
6: $\quad attsL = attsL \cup Q_{idx}$
7: **end while**
8: **return** $attsL$

Example: Query Coverage

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3$

Example: Query Coverage

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3$

1. In the first step, only queries Q1, Q3, and Q4 are considered for coverage, due to the storage constraint.

Example: Query Coverage

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3$

1. In the first step, only queries Q1, Q3, and Q4 are considered for coverage, due to the storage constraint.
2. While the same objective function value is obtained for each query, say, we choose Q1 since it uses less storage budget.

Example: Query Coverage

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3$

1. In the first step, only queries Q1, Q3, and Q4 are considered for coverage, due to the storage constraint.
2. While the same objective function value is obtained for each query, say, we choose Q1 since it uses less storage budget.
3. Now we have already chosen $\{A_1, A_2\}$. No other query can be covered in the given storage budget.

Input: Workload $W = \{Q_1, \ldots, Q_m\}$ of R; storage budget B; set of loaded attributes $saved = \{A_{s_1}, \ldots, A_{s_k}\}$

Output: Set of attributes $\{A_{s_{k+1}}, \ldots, A_{s_{k+t}}\}$ to be loaded in processing representation

1: $attsL = saved$

2: **while** $\sum_{j \in attsL} SPF_j < B$ **do**

3: $\quad idx = \text{argmax}_{j \notin attsL} \{cost(attsL) - cost(attsL \cup A_j)\}$

4: $\quad attsL = attsL \cup idx$

5: **end while**

6: **return** $attsL$

Example: Attribute Usage Frequency

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3,\ attsL = \{A_1, A_2\}$

Example: Attribute Usage Frequency

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
Q_1	X	X						
Q_2	X	X	X	X				
Q_3			X	X	X			
Q_4		X		X		X		
Q_5	X		X	X	X		X	
Q_6	X	X	X	X	X	X	X	

$B = 3$, $attsL = \{A_1, A_2\}$

A_4 is chosen as the remaining attribute to be loaded since it appears in five queries, the largest number between unloaded attributes.
$attsL = \{A_1, A_2, A_4\}$

Heuristic: Put It All Together

- Given a storage budget B, **Query coverage** is invoked first. **Attribute usage frequency** takes as input the result produced by **Query coverage** and the unused budget Δ_q.

Heuristic: Put It All Together

- Given a storage budget B, **Query coverage** is invoked first. **Attribute usage frequency** takes as input the result produced by **Query coverage** and the unused budget Δ_q.
- Instead of invoking these algorithms only once, with the given storage budget B, we consider a series of allocations. B is divided in δ increments.

Heuristic: Put It All Together

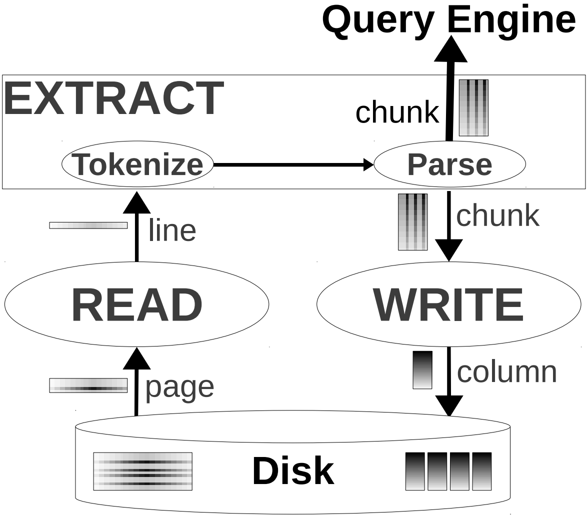
UCMERCED

- Given a storage budget B, **Query coverage** is invoked first. **Attribute usage frequency** takes as input the result produced by **Query coverage** and the unused budget Δ_q.
- Instead of invoking these algorithms only once, with the given storage budget B, we consider a series of allocations. B is divided in δ increments.
- Each algorithm is assigned anywhere from 0 to B storage, in δ increments. A solution is computed for each of these configurations. The heuristic algorithm returns the solution with the minimum objective.

Heuristic: Put It All Together

- Given a storage budget B, **Query coverage** is invoked first. **Attribute usage frequency** takes as input the result produced by **Query coverage** and the unused budget Δ_q.
- Instead of invoking these algorithms only once, with the given storage budget B, we consider a series of allocations. B is divided in δ increments.
- Each algorithm is assigned anywhere from 0 to B storage, in δ increments. A solution is computed for each of these configurations. The heuristic algorithm returns the solution with the minimum objective.
- The increment δ controls the complexity of the algorithm.

UCMERCED


Query Engine
EXTRACT
chunk
Tokenize
Parse
line
chunk
READ
WRITE
page
column
Disk

Outline

The extraction stage and reading can be overlapped:

$$T_i^{pipe} = |R| \cdot \sum_{j=1}^{n} read_{ij} \cdot \frac{SPF_j}{band_{IO}} + \max\left\{ raw_i \cdot \frac{S_{RAW}}{band_{IO}}, |R| \cdot \sum_{j=1}^{n} \left(t_{ij} \cdot T_{t_j} + p_{ij} \cdot T_{p_j} \right) \right\}$$

We have to add/modify constrains to linearize our formulation:

$$
\begin{aligned}
C_7 &: cpu_i + io_i = 1;\ \ i = \overline{1, m} \\
C_{8-10} &: cpu.x + io.x = x;\ \ x \in \{raw_i, t_{ij}, p_{ij}\} \\
C_{11-13} &: cpu.x \leq cpu_i;\ \ i = \overline{1, m} \\
C_{14-16} &: io.x \leq io_i;\ \ i = \overline{1, m}
\end{aligned}
$$

IO-Bound and CPU-Bound Threshold

UCMERCED

$$PT = \left\lceil \frac{\frac{S_{RAW}}{band_{IO}} - |R| \cdot \sum_{j=1}^{n} T_{t_j}}{\frac{|R| \cdot \sum_{j=1}^{n} T_{p_j}}{n}} \right\rceil$$

PT gives the number of attributes that can be parsed in the time required to access the raw data.

$$C_{17} : \sum_{j=1}^{n} p_{ij} - PT < cpu_i \cdot n; \;\; i = \overline{1, m}$$

$$C_{18} : PT - \sum_{j=1}^{n} p_{ij} \leq io_i \cdot n; \;\; i = \overline{1, m}$$

Linearize Objective Function

After adding $(m + n \cdot m)$ variables and $(4m + 6n \cdot m)$ constraints, we obtain the linear formulation.

$$T_i = io.raw_i \cdot \frac{S_{RAW}}{band_{IO}} + |R| \cdot \sum_{j=1}^{n} read_{ij} \cdot \frac{SPF_j}{band_{IO}} +$$
$$|R| \cdot \sum_{j=1}^{n} \left(cpu.t_{ij} \cdot T_{t_j} + cpu.p_{ij} \cdot T_{p_j} \right)$$

UCMERCED

Observation

If an IO-bound query is not covered in the **Query coverage** section of the heuristic, its contribution to the objective function cannot be improved since it cannot be completely covered by **Attribute usage frequency**.

Heuristic for Pipeline

UCMERCED

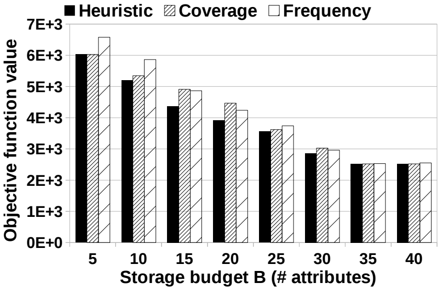
Observation

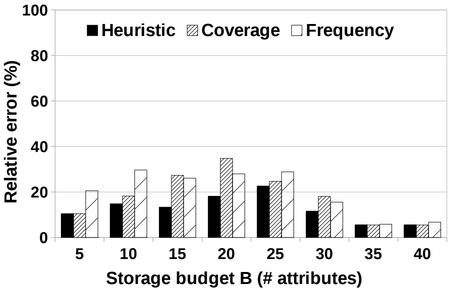
If an IO-bound query is not covered in the **Query coverage** section of the heuristic, its contribution to the objective function cannot be improved since it cannot be completely covered by **Attribute usage frequency**.

Based on this observation, the only strategy to reduce the cost is to select attributes that appear in CPU-bound queries. We enforce this by limiting the selection of the attributes considered in **Attribute usage frequency** to those attributes that appear in at least one CPU-bound query.

Outline

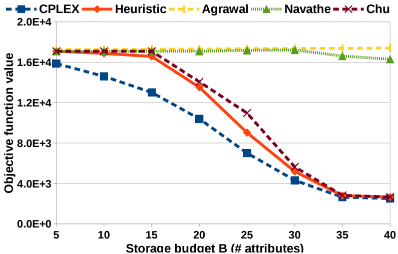
Comparison between the Two Heuristic Stages

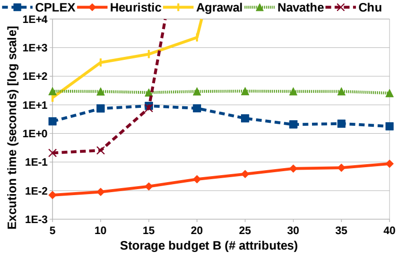




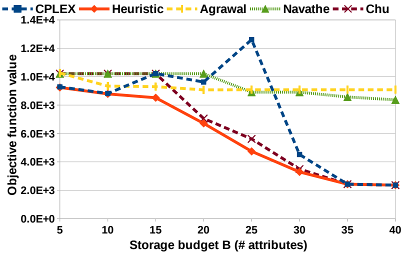
Experimental Evaluation

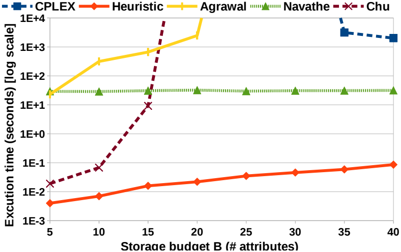
Serial

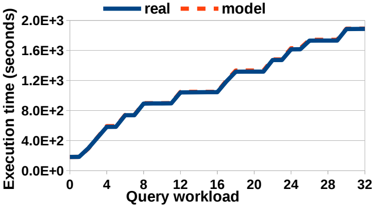




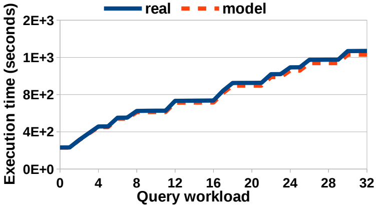
Pipelined



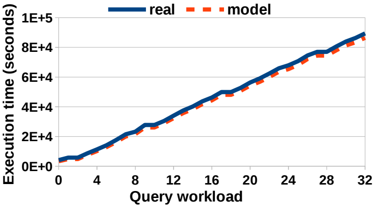




Serial CSV



Serial FITS



Pipelined JSON

Outline

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency.

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency.
- We extend the optimization formulation and the heuristic to a restricted type of pipelined raw data processing.

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency.
- We extend the optimization formulation and the heuristic to a restricted type of pipelined raw data processing.
- We evaluate the performance of the heuristic and the accuracy of the optimization formulation over three real data formats: CSV, FITS, and JSON.

Conclusions

- We study the problem of workload-driven raw data processing with partial loading.
- We provide a linear mixed integer programming optimization formulation that we prove to be NP-hard and inapproximable.
- We design a two-stage heuristic that combines the concepts of query coverage and attribute usage frequency.
- We extend the optimization formulation and the heuristic to a restricted type of pipelined raw data processing.
- We evaluate the performance of the heuristic and the accuracy of the optimization formulation over three real data formats: CSV, FITS, and JSON.
- The results confirm the superior performance of the proposed heuristic over related vertical partitioning algorithms and the accuracy of the formulation in capturing the execution details of a real operator.

Thank you!
Questions?