Vertical Partitioning for Query Processing over Raw Data

Weijie Zhao, Yu Cheng, and Florin Rusu

University of California, Merced

June 30, 2015
Outline

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

Weijie Zhao, Yu Cheng, and Florin Rusu
Vertical Partitioning for Query Processing over Raw Data
### Example SDSS Schema

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

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

| objID,skyVersion,run,rerun,camcol,field,obj,mode,nChild,type,clean,probPSF,insideMask,fl_1237645942366274027,2,109,301,3,114,491,1,0,3,1,0,0,217164284160,226,89418,0.130621,1969.1237645942366274088,2,109,301,3,114,1152,1,0,3,0,0,0,281543964623104,145.553284,0.5203641237645942366274694,2,109,301,3,114,1158,1,0,6,0,1,0,281543964623616,225.317307,0.5079171237645942370140312,2,109,301,3,173,152,1,0,6,1,1,0,15839866243776,720.658875,4.751435E1237645942370140314,2,109,301,3,173,154,1,0,6,1,1,0,68988047872,734.537415,4.991623E-3,1237645942370140321,2,109,301,3,173,161,1,0,6,1,1,0,193585182019600,776,662109,3.504038E1237645942370140322,2,109,301,3,173,162,1,0,3,0,0,0,144572653934612752,763.089349,0.59001237645942370140323,2,109,301,3,173,163,1,0,3,0,0,0,144431916446257424,797.722046,0.57421237645942370140324,2,109,301,3,173,164,1,0,3,0,0,0,316728370401624,759.693542,0.430969,1237645942370140325,2,109,301,3,173,165,1,0,3,0,0,0,144431916446257424,772.056763,0.76081237645942370140327,2,109,301,3,173,167,1,0,6,1,1,0,68987912704,813.796387,0.016672,693.1237645942370140330,2,109,301,3,173,170,1,0,6,1,1,0,35253360136208,845.047607,5.510448E1237645942370140331,2,109,301,3,173,171,1,0,3,0,0,0,105622104310104,840.993713,0.351185,1237645942370140332,2,109,301,3,173,172,1,0,6,0,1,0,387097114444048,841.738831,0.33631,1237645942370140503,2,109,301,3,173,343,1,0,6,1,1,0,68987912960,975.684814,0.06882,665.1237645942370140504,2,109,301,3,173,344,1,0,3,1,0,0,217164284160,984.625671,0.367924,97.1237645942370140505,2,109,301,3,173,345,1,0,6,1,1,0,68987912448,1008.3241,0.055251,295.51237645942370140507,2,109,301,3,173,347,1,0,3,1,0,0,68987912448,1098.70068,0.191982,131.1237645942370140509,2,109,301,3,173,349,1,0,6,1,0,0,72092847397929232,1136.47461,0.048681237645942370140510,2,109,301,3,173,350,1,0,3,1,0,0,35184640524816,1147.8905,0.054839,361237645942370140512,2,109,301,3,173,352,1,0,3,1,0,0,68988043520,1212.91077,0.191284,584.1237645942370140514,2,109,301,3,173,354,1,0,6,1,1,0,6898791292,1220.56543,0.065685,318.1237645942370140516,2,109,301,3,173,356,1,0,3,1,0,0,2278815830856,1231.65027,0.151327,781237645942370140518,2,109,301,3,173,358,1,0,3,1,0,0,35255507620624,1255.73218,0.109877,31237645942370140519,2,109,301,3,173,359,1,0,6,0,1,0,545426755424528,1270.13306,0.565565,
Example SQL Query on SDSS

\[
\begin{align*}
\text{SELECT} & \quad \text{TOP 10} \ P.\text{ObjID} \\
\text{FROM} & \quad \text{PhotoPrimary AS P JOIN Neighbors AS N ON P.ObjID} &= \ N.\text{ObjID} \\
& \quad \text{JOIN PhotoPrimary AS L ON L.ObjID} &= \ N.\text{NeighborObjID} \\
\text{WHERE} & \quad P.\text{ObjID} < L. \text{ObjID} \ \text{AND} \\
& \quad \text{abs}((P.\text{u}-P.\text{g})-(L.\text{u}-L.\text{g})) < 0.05 \ \text{AND} \\
& \quad \text{abs}((P.\text{g}-P.\text{r})-(L.\text{g}-L.\text{r})) < 0.05 \ \text{AND} \\
& \quad \text{abs}((P.\text{r}-P.\text{i})-(L.\text{r}-L.\text{i})) < 0.05 \ \text{AND} \\
& \quad \text{abs}((P.\text{i}-P.\text{z})-(L.\text{i}-L.\text{z})) < 0.05 
\end{align*}
\]
Example SQL Query on SDSS

```sql
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!
<table>
<thead>
<tr>
<th>Raw data processing with partial loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>
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

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)
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.)
Contributions

- 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.
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 Processing over Raw Data

Query Engine

EXTRACT

Tokenize

Parse

READ

line

chunk

WRITE

column

Disk

Weijie Zhao, Yu Cheng, and Florin Rusu

Vertical Partitioning for Query Processing over Raw Data
<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Suppose $B = 3$, i.e., we can load 3 attributes. What columns to choose?
Suppose $B = 3$, i.e., we can load 3 attributes. What columns to choose?
1 SDSS Data and Queries
2 Problem Statement
3 MIP Formulation
4 Heuristic Algorithm
5 Pipeline Processing
6 Experiments
7 Conclusions
Mixed Integer Programming Formulation: Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$raw_i; \ i = 0, m$</td>
<td>read raw file at query $i$</td>
</tr>
<tr>
<td>$t_{ij}; \ i = 0, m, j = 1, n$</td>
<td>tokenize attribute $j$ at query $i$</td>
</tr>
<tr>
<td>$p_{ij}; \ i = 0, m, j = 1, n$</td>
<td>parse attribute $j$ at query $i$</td>
</tr>
<tr>
<td>$read_{ij}; \ i = 1, m, j = 1, n$</td>
<td>read attribute $j$ at query $i$ from processing format</td>
</tr>
<tr>
<td>$save_j; \ j = 1, n$</td>
<td>load attribute $j$ in processing format</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>$</td>
<td>R</td>
</tr>
<tr>
<td>$S_{RAW}$</td>
<td>size of raw file</td>
</tr>
<tr>
<td>$SPF_j$, $j = 1, n$</td>
<td>size of attribute $j$ in processing format</td>
</tr>
<tr>
<td>$B$</td>
<td>size of storage in processing format</td>
</tr>
<tr>
<td>$band_{IO}$</td>
<td>storage bandwidth</td>
</tr>
<tr>
<td>$T_{t_j}$, $j = 1, n$</td>
<td>time to tokenize an instance of attribute $j$</td>
</tr>
<tr>
<td>$T_{p_j}$, $j = 1, n$</td>
<td>time to parse an instance of attribute $j$</td>
</tr>
<tr>
<td>$w_i$, $i = 1, m$</td>
<td>weight for query $i$</td>
</tr>
</tbody>
</table>
MIP Formulation

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

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

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

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

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

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

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

\[ T_{load} = \text{raw}_0 \cdot \frac{S_{RAW}}{\text{band}_{IO}} + \]

\[ |R| \cdot \sum_{j=1}^{n} \left( t_{0j} \cdot T_{t_j} + p_{0j} \cdot T_{p_j} + \text{save}_j \cdot \frac{\text{SPF}_j}{\text{band}_{IO}} \right) \]

\[ T_i = \text{raw}_i \cdot \frac{S_{RAW}}{\text{band}_{IO}} + \]

\[ |R| \cdot \sum_{j=1}^{n} \left( t_{ij} \cdot T_{t_j} + p_{ij} \cdot T_{p_j} + \text{read}_{ij} \cdot \frac{\text{SPF}_j}{\text{band}_{IO}} \right) \]
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., \( \left| \bigcup_{j=1}^{k} Q_{i_j} \right| \).
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 $W$

1: for $i = 1$ to $n$ do
2: \hspace{1em} $res = k$-element cover($W, i$)
3: \hspace{1em} if $res \geq k'$ then return $i$
4: 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 $W$

1: for $i = 1$ to $n$ do
2: \hspace{1em} $res = k$-element cover$(W, i)$
3: \hspace{1em} if $res \geq k'$ then return $i$
4: 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

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

Weijie Zhao, Yu Cheng, and Florin Rusu
Vertical Partitioning for Query Processing over Raw Data
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: $\quad idx = \arg\max_{i \notin coveredQ} \left\{ \frac{\text{cost}(attsL) - \text{cost}(attsL \cup Q_i)}{\sum_{j \in \{attsL \cup Q_i \setminus attsL\}} SPF_j} \right\}$
4: $\quad$ if $\text{cost}(attsL) - \text{cost}(attsL \cup Q_{idx}) \leq 0$ then break
5: $\quad coveredQ = coveredQ \cup idx$
6: $\quad attsL = attsL \cup Q_{idx}$
7: end while
8: return $attsL$
Example: Query Coverage

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

$B = 3$
### Example: Query Coverage

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

$B = 3$

1. In the first step, only queries $Q_1$, $Q_3$, and $Q_4$ are considered for coverage, due to the storage constraint.
Example: Query Coverage

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

$B = 3$

1. In the first step, only queries $Q_1$, $Q_3$, and $Q_4$ are considered for coverage, due to the storage constraint.
2. While the same objective function value is obtained for each query, say, we choose $Q_1$ since it uses less storage budget.
### Example: Query Coverage

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

$B = 3$

1. In the first step, only queries $Q_1$, $Q_3$, and $Q_4$ are considered for coverage, due to the storage constraint.
2. While the same objective function value is obtained for each query, say, we choose $Q_1$ 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.
Heuristic: Attribute Usage Frequency

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

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

1: $\text{attsL} = \text{saved}$
2: while $\sum_{j \in \text{attsL}} SPF_j < B$ do
3: $\text{idx} = \arg\max_{j \notin \text{attsL}} \{\text{cost}(\text{attsL}) - \text{cost}(\text{attsL} \cup A_j)\}$
4: $\text{attsL} = \text{attsL} \cup \text{idx}$
5: end while
6: return $\text{attsL}$
### Example: Attribute Usage Frequency

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_3$</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_4$</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q_5$</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

$B = 3$, $attsL = \{A_1, A_2\}$
Example: Attribute Usage Frequency

<table>
<thead>
<tr>
<th></th>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
<th>A₇</th>
<th>A₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q₁</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q₂</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q₃</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q₄</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q₅</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Q₆</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

\[ B = 3, \text{attsL} = \{A₁, A₂\} \]

\( A₄ \) is chosen as the remaining attribute to be loaded since it appears in five queries, the largest number between unloaded attributes.

\( \text{attsL} = \{A₁, A₂, A₄\} \)
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 $\Delta_q$. 
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 $\Delta q$.

Instead of invoking these algorithms only once, with the given storage budget $B$, we consider a series of allocations. $B$ is divided in $\delta$ increments.
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 $\Delta_q$.

Instead of invoking these algorithms only once, with the given storage budget $B$, we consider a series of allocations. $B$ is divided in $\delta$ increments.

Each algorithm is assigned anywhere from 0 to $B$ storage, in $\delta$ increments. A solution is computed for each of these configurations. The heuristic algorithm returns the solution with the minimum objective.
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 $\Delta_q$.

Instead of invoking these algorithms only once, with the given storage budget $B$, we consider a series of allocations. $B$ is divided in $\delta$ increments.

Each algorithm is assigned anywhere from 0 to $B$ storage, in $\delta$ increments. A solution is computed for each of these configurations. The heuristic algorithm returns the solution with the minimum objective.

The increment $\delta$ controls the complexity of the algorithm.
Query Processing over Raw Data: Pipeline Processing

EXTRACT

Tokenize

READ

Disk

WRITE

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE

Disk

EXTRACT

Tokenize

READ

Disk

WRITE
Outline

1 SDSS Data and Queries
2 Problem Statement
3 MIP Formulation
4 Heuristic Algorithm
5 Pipeline Processing
6 Experiments
7 Conclusions
The extraction stage and reading can be overlapped:

\[ T_i^{\text{pipe}} = |R| \cdot \sum_{j=1}^{n} \text{read}_{ij} \cdot \frac{\text{SPF}_j}{\text{band}_{\text{IO}}} + \]

\[
\max \left\{ \text{raw}_i \cdot \frac{S_{\text{RAW}}}{\text{band}_{\text{IO}}}, |R| \cdot \sum_{j=1}^{n} \left( \text{t}_{ij} \cdot T_{tj} + p_{ij} \cdot T_{pj} \right) \right\}
\]
We have to add/modify constraints to linearize our formulation:

\[
\begin{align*}
C_7 : \quad & cpu_i + io_i = 1; \quad i = 1, m \\
C_{8-10} : \quad & cpu.x + io.x = x; \quad x \in \{ raw_i, t_{ij}, p_{ij} \} \\
C_{11-13} : \quad & cpu.x \leq cpu_i; \quad i = 1, m \\
C_{14-16} : \quad & io.x \leq io_i; \quad i = 1, m
\end{align*}
\]
IO-Bound and CPU-Bound Threshold

\[ PT = \left[ \frac{S_{RAW \ band_{IO}} - |R| \cdot \sum_{j=1}^{n} T_{tj}}{|R| \cdot \sum_{j=1}^{n} T_{pj}} \right] \]

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

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

\[ C_{18} : PT - \sum_{j=1}^{n} p_{ij} \leq io_i \cdot n; \ i = 1, m \]
After adding \((m + n \cdot m)\) variables and \((4m + 6n \cdot m)\) constraints, we obtain the linear formulation.

\[
T_i = \text{io.raw}_i \cdot \frac{S_{\text{RAW}}}{\text{band}_{\text{IO}}} + |R| \cdot \sum_{j=1}^{n} \text{read}_{ij} \cdot \frac{\text{SPF}_j}{\text{band}_{\text{IO}}} + |R| \cdot \sum_{j=1}^{n} \left( \text{cpu.t}_{ij} \cdot T_{t_j} + \text{cpu.p}_{ij} \cdot T_{p_j} \right)
\]
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

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

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

Objective function value

Storage budget B (# attributes)

Relative error (%)

Storage budget B (# attributes)
Experimental Evaluation

Serial

Pipeli ned

Weijie Zhao, Yu Cheng, and Florin Rusu

Vertical Partitioning for Query Processing over Raw Data
Model Validation

Serial CSV

Serial FITS

Pipelined JSON

Weijie Zhao, Yu Cheng, and Florin Rusu

Vertical Partitioning for Query Processing over Raw Data
Outline

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

Weijie Zhao, Yu Cheng, and Florin Rusu
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.

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.

Weijie Zhao, Yu Cheng, and Florin Rusu
Vertical Partitioning for Query Processing over Raw Data
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 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.

Weijie Zhao, Yu Cheng, and Florin Rusu
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?