Similarity Join over Array Data

Weijie Zhao\textsuperscript{1}, Florin Rusu\textsuperscript{1,2}, Bin Dong\textsuperscript{2}, Kesheng Wu\textsuperscript{2}

University of California, Merced\textsuperscript{1}
Lawrence Berkeley National Laboratory\textsuperscript{2}

June 30, 2016
Outline

1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions
Astronomical Catalogs

- Sloan Digital Sky Survey (SDSS), Palomar Transient Factory (PTF), etc.
- Transient identification: supernovae detection, exoplanet search
Array Construction

- Image $\xrightarrow{\text{Image processing}}$ Objects $\rightarrow$ 3-D array $[\text{ra, dec, time}]$
Array Construction

- Image processing $\rightarrow$ Objects $\rightarrow$ 3-D array $[ra, dec, time]$
- Array is chunked and stored in a shared-nothing architecture

Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu

Similarity Join over Array Data
Array Construction

- Image $\xrightarrow{\text{Image processing}}$ Objects $\rightarrow$ 3-D array $[\text{ra}, \text{dec}, \text{time}]$
- Array is chunked and stored in a shared-nothing architecture
- Array is sparse and skewed
Detect an object: Does it exist in a catalog?

Coordinates for the same object in different catalogs may not be EXACTLY the same

Find object’s “neighbors” in the catalog

→ Join each cell with “neighbor cells” in catalog
Contributions

- Formal array similarity join definition
Contributions

- Formal array similarity join definition
- Array similarity join operator – data transfer, network, load balancing
Contributions

- Formal array similarity join definition
- Array similarity join operator – data transfer, network, load balancing
- Modeling data transfer as vertex cover problem
 Contributions

- Formal array similarity join definition
- Array similarity join operator – data transfer, network, load balancing
- Modeling data transfer as vertex cover problem
- Experimentally compare against existing solutions on PTF catalog and LinkedGeoData
Related Work

- Relational similarity join: $\epsilon - kdB$ tree, EGO family, etc.
  - Proposed operator does not require repartitioning and heavy preprocessing
- MapReduce similarity join: MRSimJoin, ClusterJoin, MR-DSJ, PHiDJ, etc.
  - Proposed operator takes advantage of array chunking to schedule join processing and reduce data transfer
- Array equi-join: structural join (Soroush 2011), shuffle join (Duggan 2015)
  - Proposed operator overlaps communication and computation
  - More complicated MIP optimization formulation
  - Graph formulation and heuristic solution
  - Generalization to more general joins
Outline

1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions
Array Data Model

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>(3, 4)</td>
<td></td>
<td></td>
<td>(3, 3)</td>
<td></td>
<td></td>
<td>(8, 8)</td>
<td>(3, 1)</td>
</tr>
<tr>
<td>[2]</td>
<td></td>
<td>(9, 8)</td>
<td>(4, 4)</td>
<td>(9, 3)</td>
<td></td>
<td></td>
<td></td>
<td>(2, 4)</td>
</tr>
<tr>
<td>[3]</td>
<td>(2, 3)</td>
<td>(6, 6)</td>
<td></td>
<td>(1, 0)</td>
<td>(1, 1)</td>
<td></td>
<td>(4, 3)</td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>(1, 5)</td>
<td>(9, 9)</td>
<td>(4, 5)</td>
<td>(7, 8)</td>
<td>(2, 4)</td>
<td>(3, 9)</td>
<td>(2, 8)</td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td></td>
<td>(8, 5)</td>
<td>(3, 8)</td>
<td></td>
<td>(1, 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td></td>
<td>(5, 5)</td>
<td></td>
<td>(2, 5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td>(3, 4)</td>
<td>(7, 1)</td>
<td>(2, 2)</td>
<td></td>
<td></td>
<td>(6, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8]</td>
<td>(7, 8)</td>
<td>(3, 2)</td>
<td></td>
<td>(8, 1)</td>
<td>(7, 5)</td>
<td>(7, 7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

server X server Y server Z

Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu

Similarity Join over Array Data
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>7</td>
<td></td>
<td></td>
<td>18</td>
<td></td>
<td></td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>[2]</td>
<td></td>
<td>37</td>
<td>37</td>
<td>27</td>
<td></td>
<td></td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>23</td>
<td>34</td>
<td></td>
<td>24</td>
<td>18</td>
<td></td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td></td>
<td></td>
<td>52</td>
<td>33</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td></td>
<td></td>
<td>27</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td>30</td>
<td>24</td>
<td>22</td>
<td></td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8]</td>
<td>27</td>
<td>28</td>
<td>21</td>
<td>35</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shape: [(7, 8), (2, 4), (3, 9), (1, 5)]
SELECT $f$ INTO $\tau$ FROM $\alpha$ SIMILARITY JOIN $\beta$ ON $M$
WITH SHAPE $\sigma$
SELECT f INTO $\tau$ FROM $\alpha$ SIMILARITY JOIN $\beta$ ON $\mathcal{M}$ WITH SHAPE $\sigma$

$f$ : computation function, $\tau$ : result array, $\alpha, \beta$ : input arrays, $\mathcal{M}$ : array mapping function, $\sigma$ : similarity shape array
SELECT \( f \) INTO \( \tau \) FROM \( \alpha \) SIMILARITY JOIN \( \beta \) ON \( \mathcal{M} \) WITH SHAPE \( \sigma \)

\( f \): computation function, \( \tau \): result array, \( \alpha, \beta \): input arrays, \( \mathcal{M} \): array mapping function, \( \sigma \): similarity shape array
Similarity Shape Array for Popular Distance Metrics

$L^0$  $L^1$  $L^2$  $L^\infty$  EMD  Hamming

Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu
Outline

1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions
Find all chunk pairs that have to be joined
For each chunk pair \((\Upsilon, \Psi)\), \(\Upsilon \in \alpha, \Psi \in \beta\), \(\Upsilon\) and \(\Psi\) must be transferred to the same node
Compute \(f\) for each chunk pair transferred to the same node
Overlap I/O (network and disk) and processing through multi-threading
Problem Formulation

\[
\begin{align*}
\min \{ & \\
\max_{ijk} \{ & y_{ijkt} \cdot t \cdot T_{ntwk} \}, \\
\max_k \{ & \sum_{tij} y_{ijk(t-1)} \sum_{i' \in \Psi_i} (1 - z_{i'j(t-1)}) \cdot T_{disk} \} \\
\} \end{align*}
\]

\[C_1: \ x_{ijk} + x_{i'kj} = 1; \forall i, \forall i' \in \Psi_i, p(i) = j, p(i') = k\]

\[C_2: \ \sum_t y_{ijkt} = x_{ijk}; \forall i, \forall j, \forall k\]

\[C_3: \ \sum_{ijk} y_{ijkt} = 1; \forall t\]

\[C_4: \ \sum_{ik} y_{ijkt} = 1; \forall t, \forall j\]

\[C_5: \ \sum_{ij} y_{ijkt} = 1; \forall t, \forall k\]

\[C_6: \ \sum_i z_{ijt} \leq B; \forall t, \forall j\]

\[C_7: \ z_{i'jt} \leq z_{i'j(t-1)} + y_{ikjt}; \forall t, \forall k, \forall j, \forall i' \in \Psi_i\]
Decompose the optimization formulation into 3 independent stages

Each stage takes the optimization output of the previous stage as a pre-condition
Consider chunk pair \((\Upsilon, \Psi)\), \(\Upsilon\) on Node \(X\), \(\Psi\) on Node \(Y\) with 
\(\text{size}(\Upsilon) = 3\), \(\text{size}(\Psi) = 2\)

No! If we have another chunk pair \((\Upsilon', \Psi')\), \(\Psi'\) on Node \(Y\) with 
\(\text{size}(\Psi') = 2\). \(\Upsilon' : X \rightarrow Y\) (cost 3) is better than 
\(\Psi, \Psi' : Y \rightarrow X\) (cost 2 + 2 = 4)
Consider chunk pair \((\Upsilon, \Psi)\), \(\Upsilon\) on Node X, \(\Psi\) on Node Y with
\[\text{size}(\Upsilon) = 3, \text{size}(\Psi) = 2\]
Transfer the smaller chunk to the larger chunk’s node?
\((\Psi : Y \rightarrow X)\)
Consider chunk pair \((\Upsilon, \Psi)\), \Upsilon\ on Node X, \Psi on Node Y with \(\text{size}(\Upsilon) = 3\), \(\text{size}(\Psi) = 2\)

Transfer the smaller chunk to the larger chunk’s node? \((\Psi : Y \rightarrow X)\)

No! If we have another chunk pair \((\Upsilon, \Psi')\), \Psi'\ on Node Y with \(\text{size}(\Psi') = 2\). \(\Upsilon : X \rightarrow Y\) (cost 3) is better than \(\Psi, \Psi' : Y \rightarrow X\) (cost 2 + 2 = 4)
Consider chunk pair \((\Upsilon, \Psi)\), \(\Upsilon\) on Node X, \(\Psi\) on Node Y with size(\(\Upsilon\)) = 3, size(\(\Psi\)) = 2

Transfer the smaller chunk to the larger chunk’s node? (\(\Psi : Y \rightarrow X\))

No! If we have another chunk pair \((\Upsilon, \Psi')\), \(\Psi'\) on Node Y with size(\(\Psi'\)) = 2. \(\Upsilon : X \rightarrow Y\) (cost 3) is better than \(\Psi, \Psi' : Y \rightarrow X\) (cost 2 + 2 = 4)
Consider chunk pair \((\Upsilon, \Psi)\), \(\Upsilon\) on Node X, \(\Psi\) on Node Y with size(\(\Upsilon\)) = 3, size(\(\Psi\)) = 2.

Transfer the smaller chunk to the larger chunk’s node? (\(\Psi : Y \rightarrow X\))

No! If we have another chunk pair \((\Upsilon, \Psi')\), \(\Psi'\) on Node Y with size(\(\Psi'\)) = 2. \(\Upsilon : X \rightarrow Y\) (cost 3) is better than \(\Psi, \Psi' : Y \rightarrow X\) (cost 2 + 2 = 4).
Transfer Schedule Optimization

- Avoid many nodes talking to same receiver node at the same time

In addition, even though we know two nodes will communicate in a period of time, what chunks in what order should be transferred?

Chunk order: choose chunk from the same sender node that shares the largest number of local chunks with the receiver.
Transfer Schedule Optimization

- Avoid many nodes talking to same receiver node at the same time
- Node order: enforce strict node-to-node communication at any instant in time in order to minimizes network congestion

![Graph example]
Transfer Schedule Optimization

- Avoid many nodes talking to same receiver node at the same time
- Node order: enforce strict node-to-node communication at any instant in time in order to minimize network congestion

In addition, even though we know two nodes will communicate in a period of time, what chunks in what order should be transferred?
Transfer Schedule Optimization

- Avoid many nodes talking to the same receiver node at the same time.
- Node order: enforce strict node-to-node communication at any instant in time in order to minimize network congestion.

In addition, even though we know two nodes will communicate in a period of time, what chunks in what order should be transferred?
- Chunk order: choose chunk from the same sender node that shares the largest number of local chunks with the receiver.
Which chunk should be kicked out when memory budget is full?
Disk Access Plan Optimization

- Which chunk should be kicked out when memory budget is full?
- The one that will be used in the furthest future?
Disk Access Plan Optimization

- Which chunk should be kicked out when memory budget is full?
- The one that will be used in the furthest future?
- According to chunk access plan, we know the future!

Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu
Outline

1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions

Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu

Similarity Join over Array Data
Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu

Experimental Evaluation

Execution time:
PTF 3D ($L^\infty$)  
PTF 2D ($L^1$)  
LinkedGeoData ($L^2$)

Data transferred:
PTF 3D ($L^\infty$)  
PTF 2D ($L^1$)  
LinkedGeoData ($L^2$)
Outline

1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions
Conclusions

- We define formally array similarity join with a shape array
We define formally array similarity join with a shape array

We introduce the first array similarity join operator that minimizes the overall data transfer and network congestion while providing load-balancing in a multi-thread pipelined architecture
Conclusions

- We define formally array similarity join with a shape array.
- We introduce the first array similarity join operator that minimizes the overall data transfer and network congestion while providing load-balancing in a multi-thread pipelined architecture.
- We model the query optimization of array similarity join as a vertex cover problem and introduce efficient algorithms to find optimal execution plans.
Conclusions

- We define formally array similarity join with a shape array.
- We introduce the first array similarity join operator that minimizes the overall data transfer and network congestion while providing load-balancing in a multi-thread pipelined architecture.
- We model the query optimization of array similarity join as a vertex cover problem and introduce efficient algorithms to find optimal execution plans.
- The proposed array similarity join operator is running inside the PTF pipeline at Lawrence Berkeley National Lab’s NERSC.
Thank you!
Questions?