Similarity Join over Array Data

Weijie Zhao[1], Florin Rusu[1,2], Bin Dong[2], Kesheng Wu[2]

University of California, Merced[1]
Lawrence Berkeley National Laboratory[2]

June 30, 2016

Outline

UCMERCED

Astronomical Catalogs

UCMERCED

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

Array Construction

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

Array Construction

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

Array Construction

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

Cross-Matching

- 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
- $\rightarrow$ 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

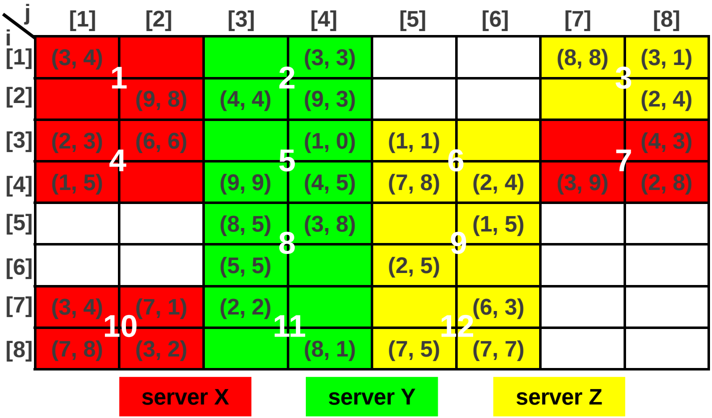
- Relational similarity join: $\epsilon - kdB$ tree, EGO family, etc.
  - Proposed operator does not require repartitioning and heavy preprocessing
- MapReduce similarity join: MRSimJoin, ClusterJoin, MR-DSJ, PHiDJ, etc.
  - Proposed operator takes advantage of array chunking to schedule join processing 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

UCMERCED

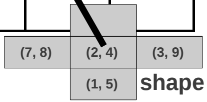
1. Cross-Matching Astronomical Catalogs
2. Array Similarity Join
3. Query Optimization
4. Experimental Evaluation
5. Conclusions

Array Data Model



APPLY operator (Baumann 1998, Marathe 2002)

i \ j	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
[1]	7			18			20	26
[2]		37	37	27				17
[3]	23	34		24	18			23
[4]	11		40	56	32	39	28	29
[5]			52	33		12		
[6]			27		7			
[7]	30	24	22			23		
[8]	27	28		21	35	35		



Array Similarity Join

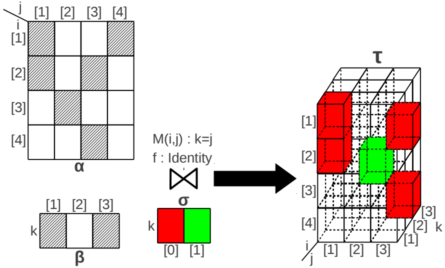
- SELECT f INTO τ FROM α SIMILARITY JOIN β ON $\mathcal{M}$ WITH SHAPE σ

- SELECT f INTO τ FROM α SIMILARITY JOIN β ON $\mathcal{M}$ WITH SHAPE σ

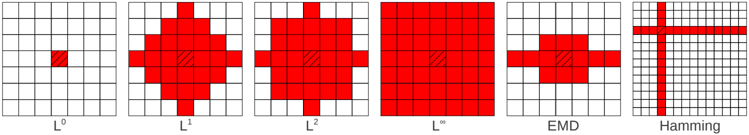
f : computation function, τ : result array, α, β : input arrays, $\mathcal{M}$: array mapping function, σ : similarity shape array

- SELECT f INTO τ FROM α SIMILARITY JOIN β ON $\mathcal{M}$ WITH SHAPE σ

f : computation function, τ : result array, α, β : input arrays, $\mathcal{M}$: array mapping function, σ : similarity shape array



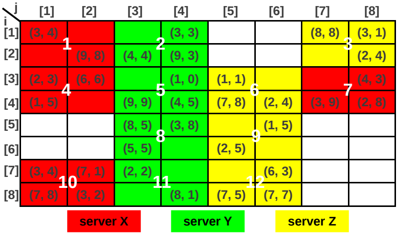
Similarity Shape Array for Popular Distance Metrics



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 $(\Upsilon, \Psi), \Upsilon \in \alpha, \Psi \in \beta$, Υ and Ψ 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

$$\min \Big\{ \max_{ijk} \{ y_{ijkt} \cdot t \cdot T_{ntwk} \}, \max_{k} \{ \sum_{tij} y_{ijk(t-1)} \sum_{i' \in \Psi_i} \big(1 - z_{i'j(t-1)}\big) \cdot T_{disk} \} \Big\}$$

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

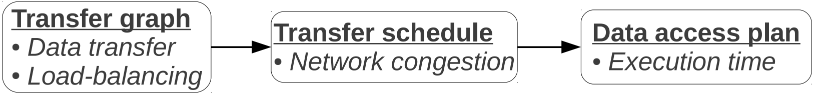
$$C_2 : \sum_{t} y_{ijkt} = x_{ijk}; \forall i, \forall j, \forall k \qquad C_3 : \sum_{ijk} y_{ijkt} = 1; \forall t$$

$$C_4 : \sum_{ik} y_{ijkt} = 1; \forall t, \forall j \qquad C_5 : \sum_{ij} y_{ijkt} = 1; \forall t, \forall k$$

$$C_6 : \sum_{i} z_{ijt} \leq B; \forall t, \forall j$$

$$C_7 : z_{i'jt} \leq z_{i'j(t-1)} + y_{ikjt}; \forall t, \forall k, \forall j, \forall i' \in \Psi_i$$

Query Optimization



- Decompose the optimization formulation into 3 independent stages
- Each stage takes the optimization output of the previous stage as a pre-condition

Transfer Graph

- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ on Node Y with size$(\Upsilon) = 3$, size$(\Psi) = 2$

Transfer Graph

- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ on Node Y with size$(\Upsilon) = 3$, size$(\Psi) = 2$
- Transfer the smaller chunk to the larger chunk's node? $(\Psi : Y \rightarrow X)$

Transfer Graph

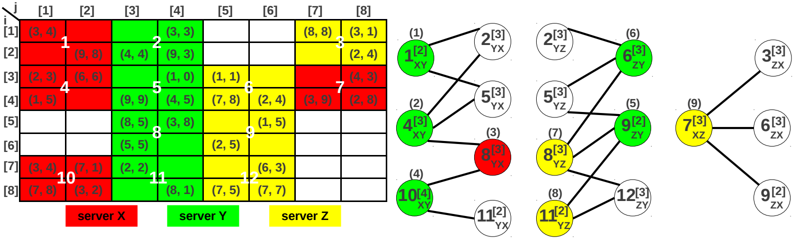
UCMERCED

- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ 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 (Υ, Ψ'), Ψ' 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 Graph

UCMERCED

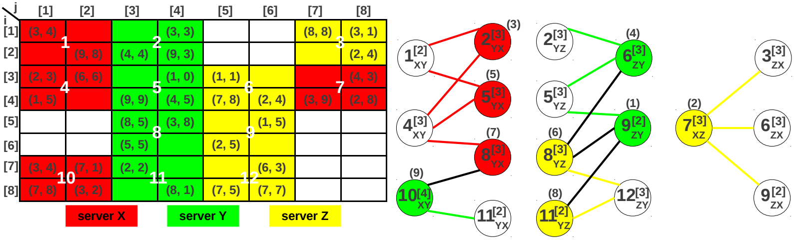
- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ on Node Y with $\text{size}(\Upsilon) = 3$, $\text{size}(\Psi) = 2$
- Transfer the smaller chunk to the larger chunk's node? $(\Psi : Y \rightarrow X)$
- No! If we have another chunk pair (Υ, Ψ'), Ψ' on Node Y with $\text{size}(\Psi') = 2$. $\Upsilon : X \rightarrow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



Transfer Graph

UCMERCED

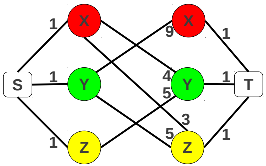
- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ on Node Y with size$(\Upsilon) = 3$, size$(\Psi) = 2$
- Transfer the smaller chunk to the larger chunk's node? $(\Psi : Y \rightarrow X)$
- No! If we have another chunk pair (Υ, Ψ'), Ψ' on Node Y with size$(\Psi') = 2$. $\Upsilon : X \rightarrow Y$ (cost 3)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



Transfer Schedule Optimization

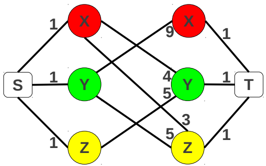
- Avoid many nodes talking to same receiver node at the same time

- 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



Transfer Schedule Optimization

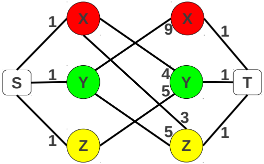
- Avoid many nodes talking to same receiver node at the same time
- Node order: enforce strict node-to-node communication at any instant in time in order to minimizes network congestion



- 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 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



- 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

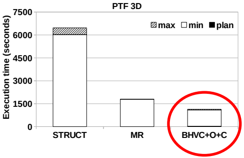
- Which chunk should be kicked out when memory budget is full?
- The one that will be used in the furthest future?
- According to chunk access plan, we know the future!

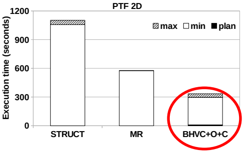
Outline

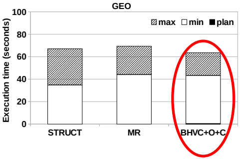
Experimental Evaluation

Execution time:

PTF 3D (L^∞) PTF 2D (L^1) LinkedGeoData (L^2)

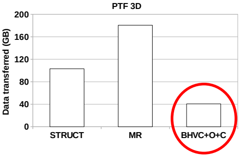


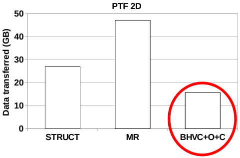


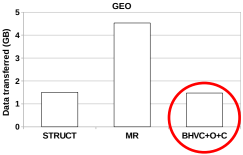


Data transferred:

PTF 3D (L^∞) PTF 2D (L^1) LinkedGeoData (L^2)







Outline

UCMERCED

Conclusions

- We define formally array similarity join with a shape array

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

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?