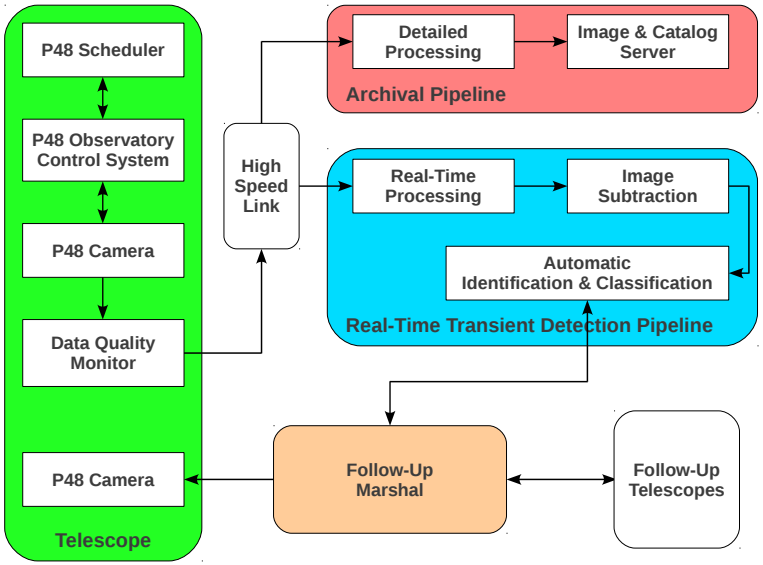
Advanced Database Techniques for Processing Scientific Multi-Dimensional Data

Florin Rusu

University of California Merced
Lawrence Berkeley National Laboratory

Palomar Transient Factory (PTF)

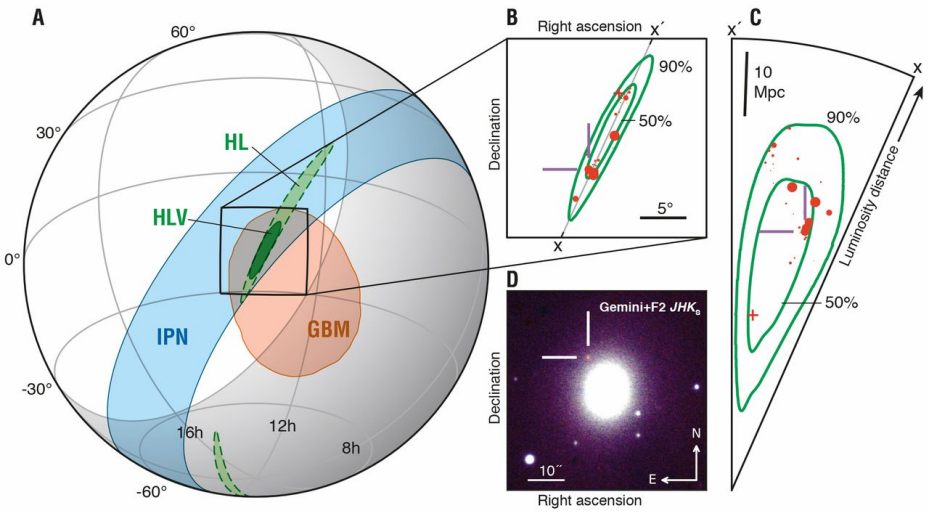


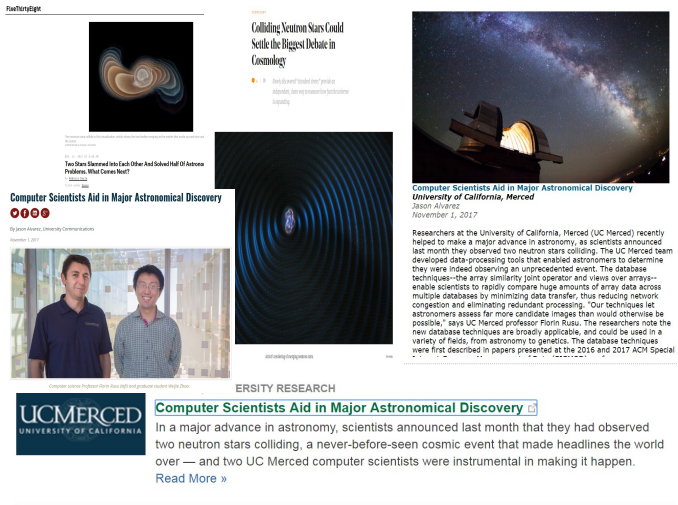
- 2,000-4,000 images 2,048x4,096 pixels per night
- 60-100 GB per night
- Complete processing of an image set within 30-45 minutes of aquisition

Automatic Transient Identification & Classification

- 1-1.5 million candidates extracted per night; 10,000 every 45 minutes; 30-150 are real
- Real-or-bogus classification: Is the candidate real?
- What is the transient type of a real candidate: VarStar? SN/Nova? circumnuclear event? asteroid?

First-Ever Neutron Star Merger Observation



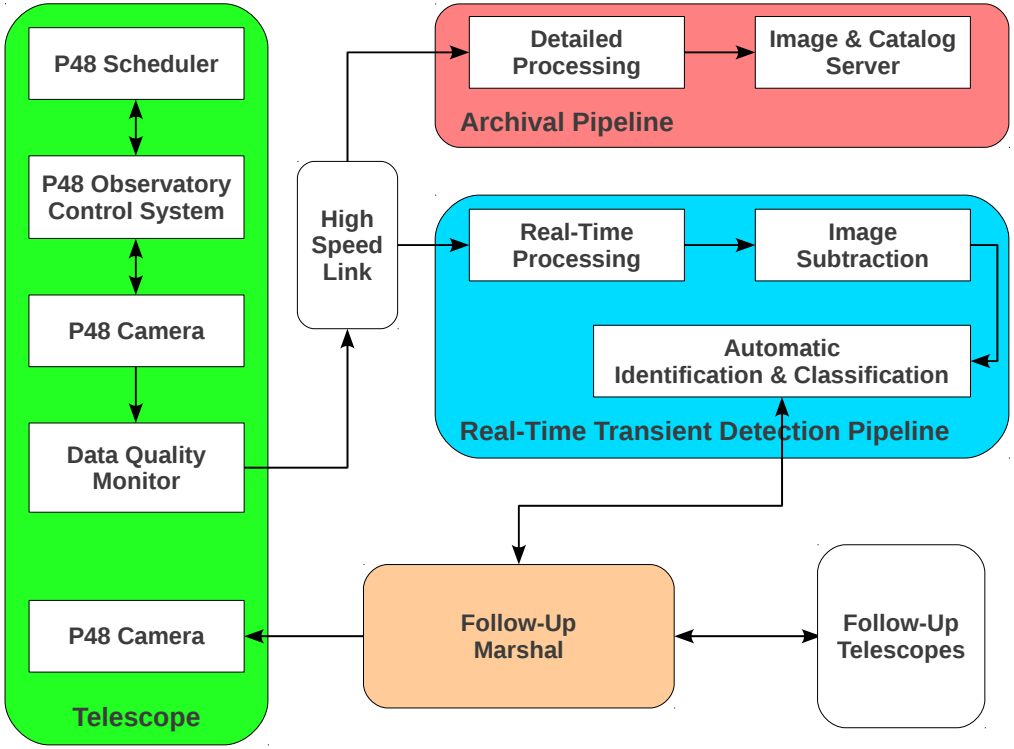


- Produces gravitational waves and turns out to be the origin of heavy elements, including gold
- **Science** article: *Illuminating gravitational waves: A concordant picture of photons from a neutron star merger*

Outline

1. Array Similarity Join [SIGMOD 2016]
2. Array Views [SIGMOD 2017]
3. Distributed Caching of Raw Arrays [arXiv 2018]
4. ArrayUDF [HPDC 2017]
5. Collaborators

Array Construction



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

Cross-Matching Astronomical Catalogs

- PTF, Sloan Digital Sky Survey (SDSS), etc.
- 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

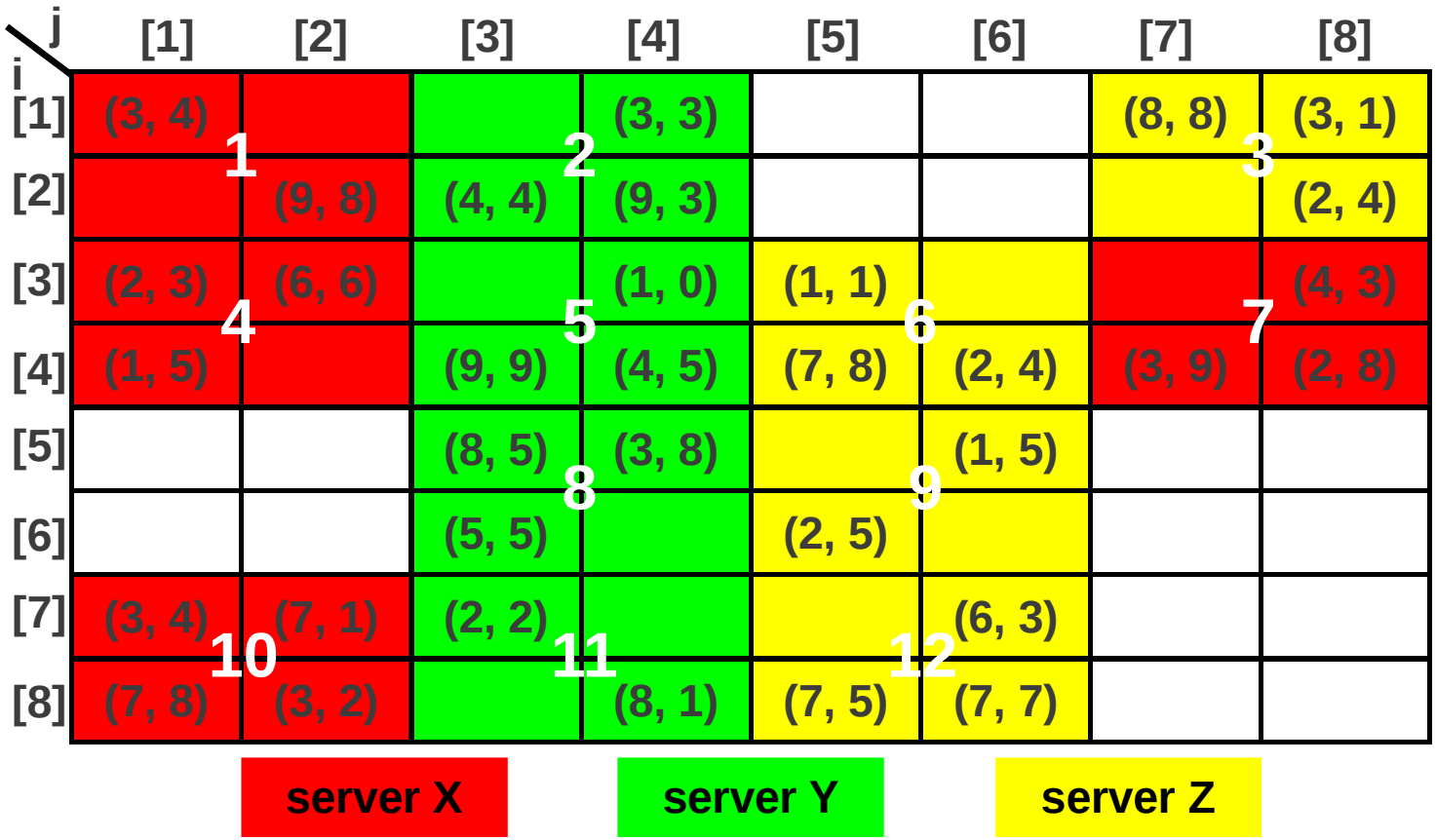
Contributions

- Formal array similarity join definition
- 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

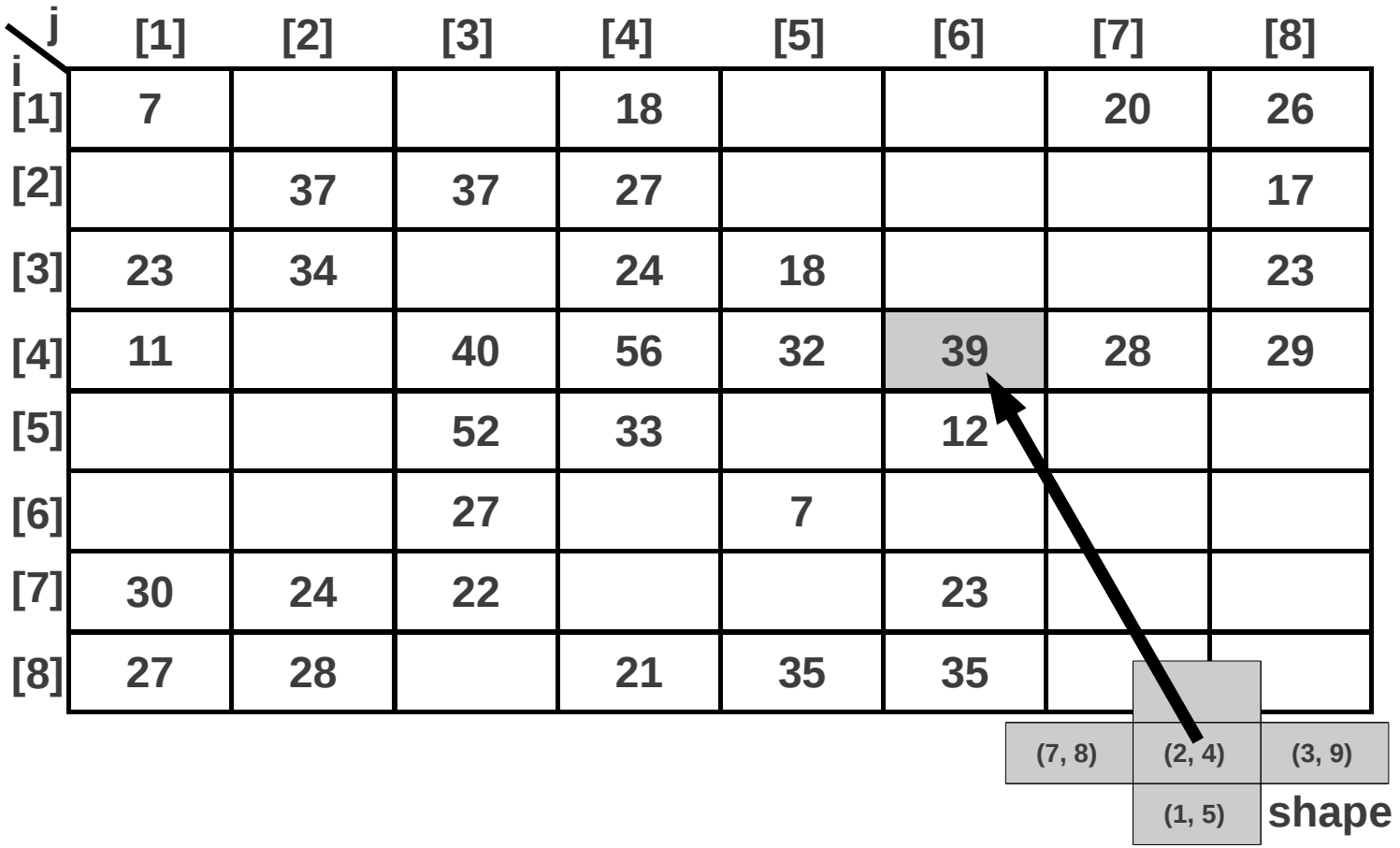
Related Work

- 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

Array Data Model



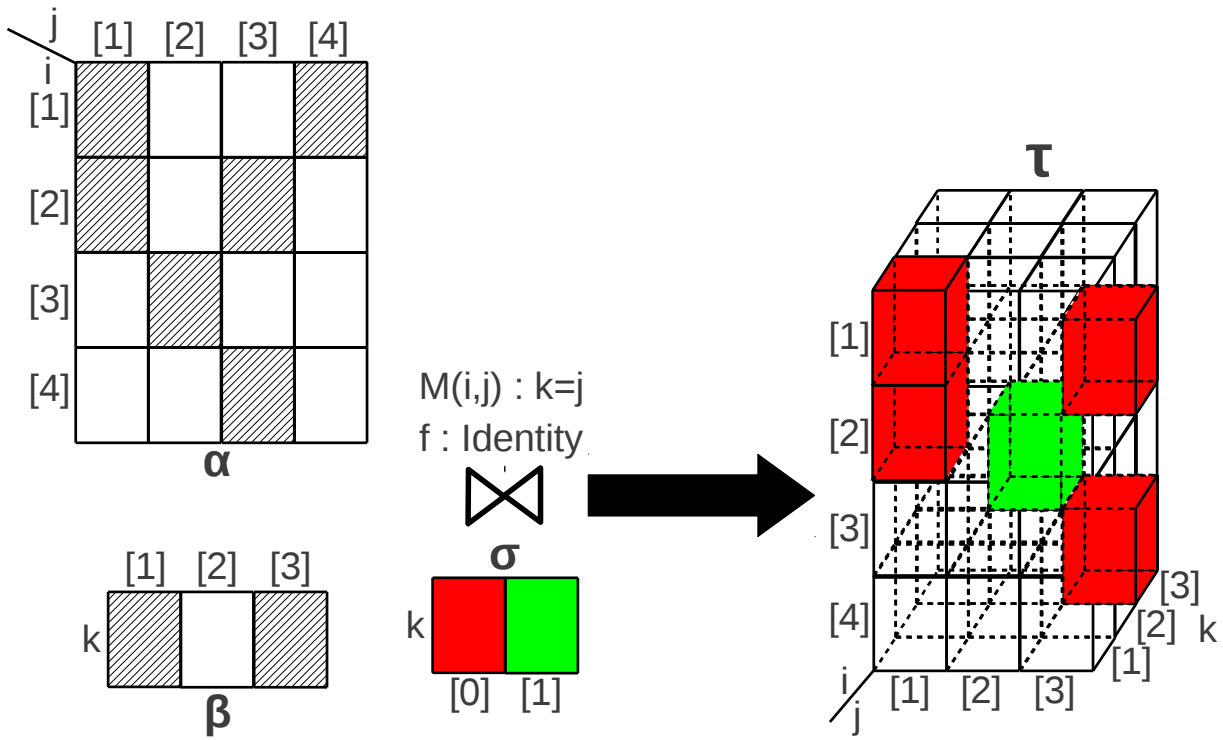
APPLY operator (Baumann 1998, Marathe 2002)



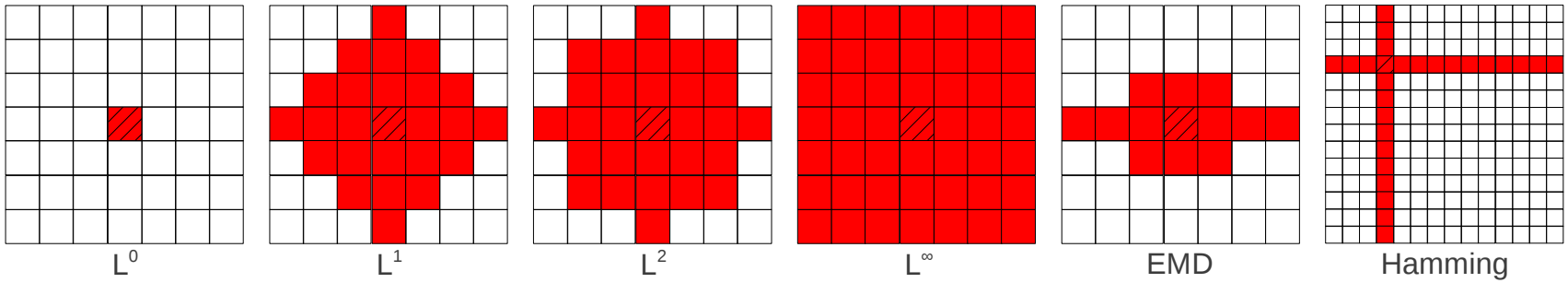
Array Similarity Join

- 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

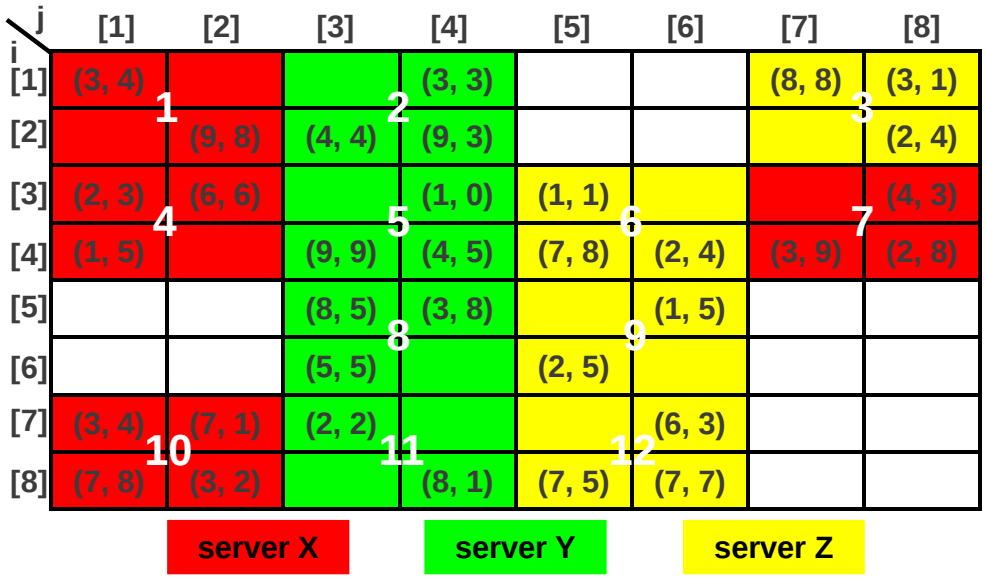


Similarity Shape Array for Popular Distance Metrics



Query Execution

- 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

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

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

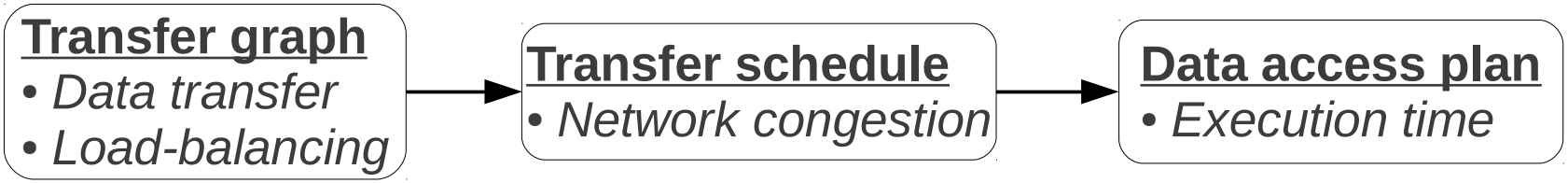
$$C_2 : \sum_{t} y_{ijkt} = x_{ijk}; \forall i, \forall j, \forall k \qquad C_3 : \sum_{ijk} y_{ijkt} = 1; \forall t$$

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

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

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

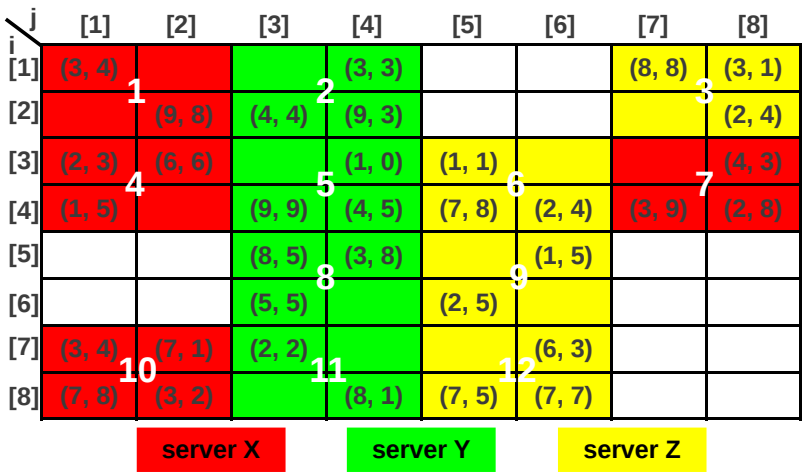
Query Optimization

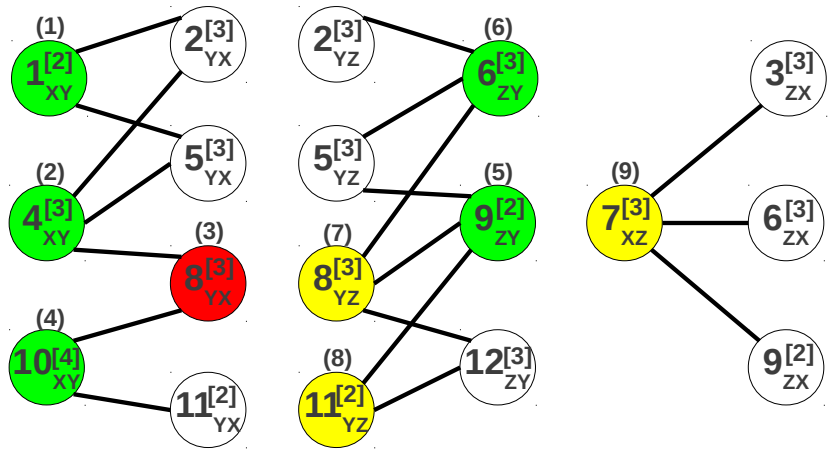


- Decompose the optimization formulation into 3 independent stages
- Each stage takes the optimization output of the previous stage as a pre-condition

Transfer Graph

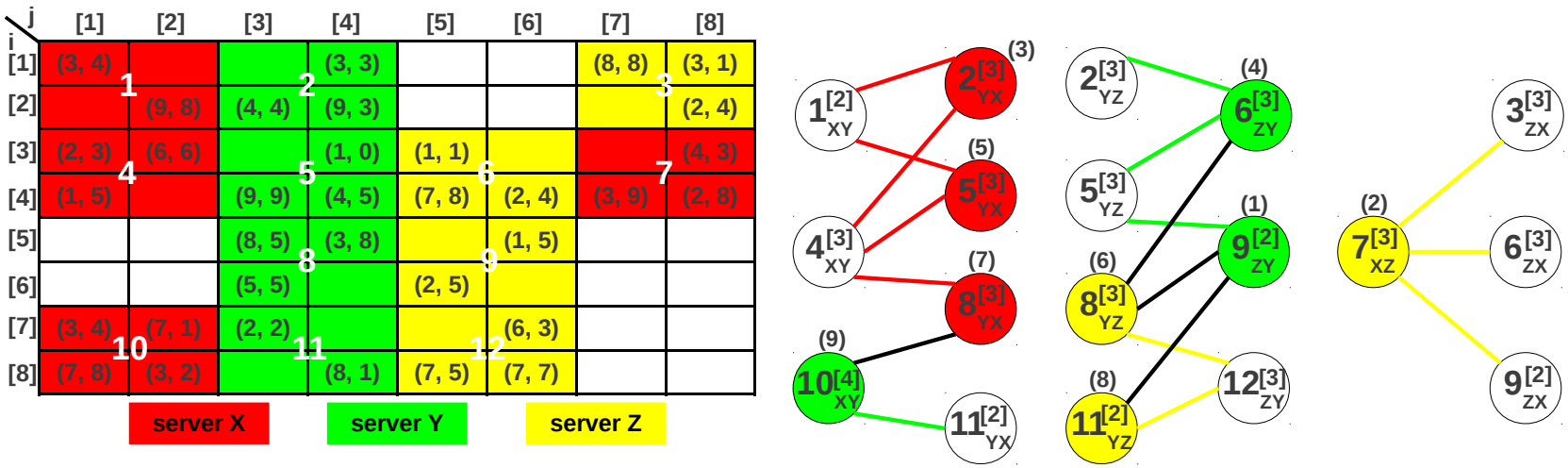
- Consider chunk pair (Υ, Ψ), Υ on Node X, Ψ on Node Y with $\text{size}(\Upsilon) = 3$, $\text{size}(\Psi) = 2$
- Transfer the smaller chunk to the larger chunk's node? $(\Psi : Y \rightarrow X)$
- No! If we have another chunk pair (Υ, Ψ'), Ψ' on Node Y with $\text{size}(\Psi') = 2$. $\Upsilon : X \rightarrow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





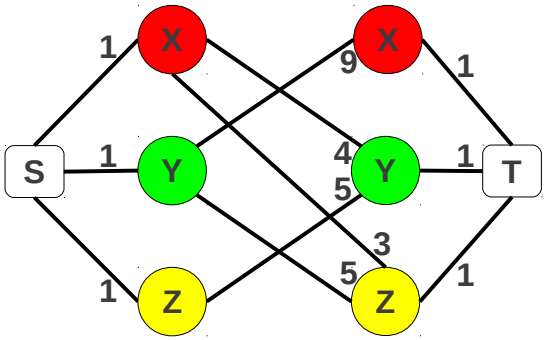
Transfer Graph

- 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 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

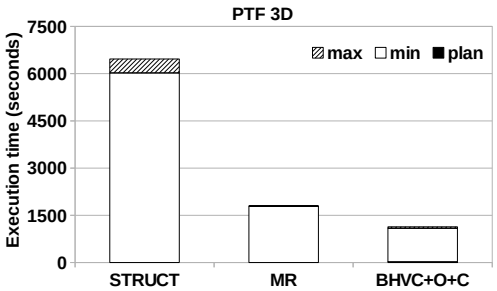
Disk Access Plan Optimization

- 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!

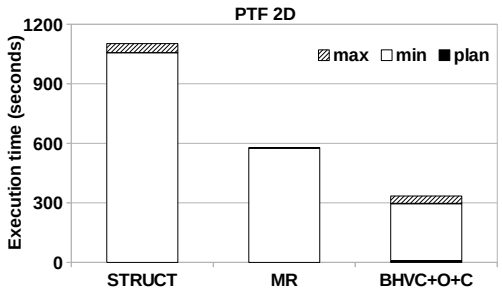
Experimental Evaluation

Execution time:

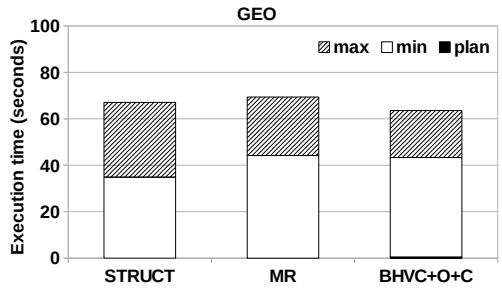
PTF 3D (L^∞)



PTF 2D (L^1)

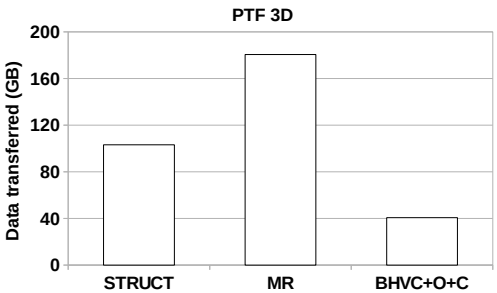


LinkedGeoData (L^2)

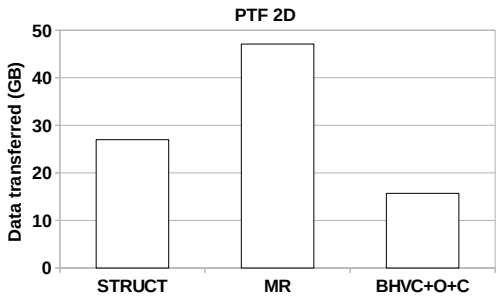


Data transferred:

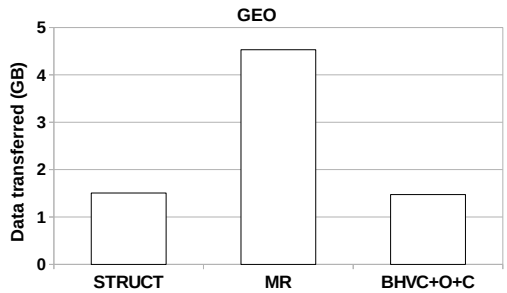
PTF 3D (L^∞)



PTF 2D (L^1)



LinkedGeoData (L^2)



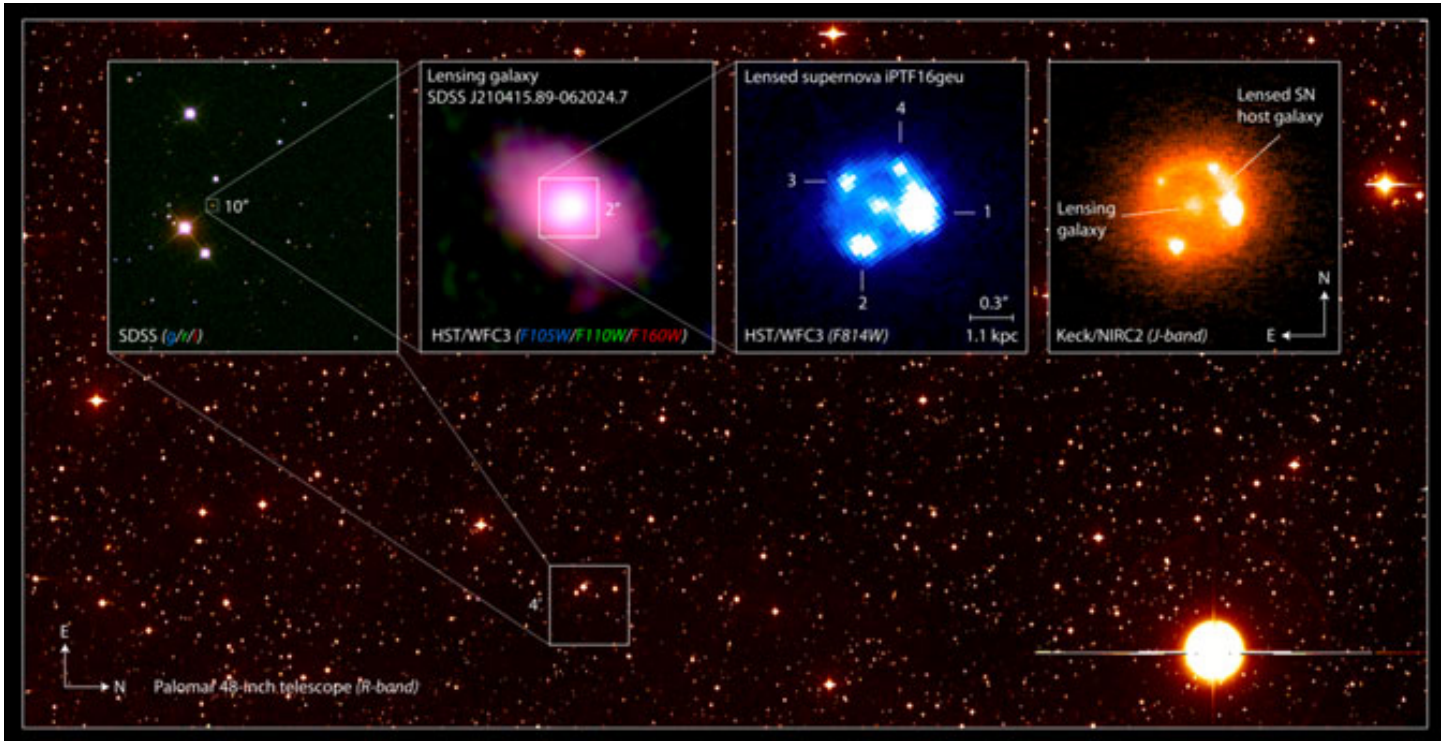
Recap

- 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

Outline

1. Array Similarity Join [SIGMOD 2016]
2. Array Views [SIGMOD 2017]
3. Distributed Caching of Raw Arrays [arXiv 2018]
4. ArrayUDF [HPDC 2017]
5. Collaborators

Astronomical Data Products



- Gamma Ray Burst: extremely energetic explosions in distant galaxies that destroy everything around
  - For each dectection, all its neighbor detections are within 12 hours with increasing magnitude
- M-Dwarf Flares: periodic flares with period larger than a month
  - No two neighbor dectections within a month

Astronomical Data Products

- Neighborhood matching $\rightarrow$ Array similarity join (expensive)
- Astronomical data products $\rightarrow$ Materialized query result
- New detections every night $\rightarrow$ Base array updates
- Maintained data products $\rightarrow$ Incremental view maintenance over arrays

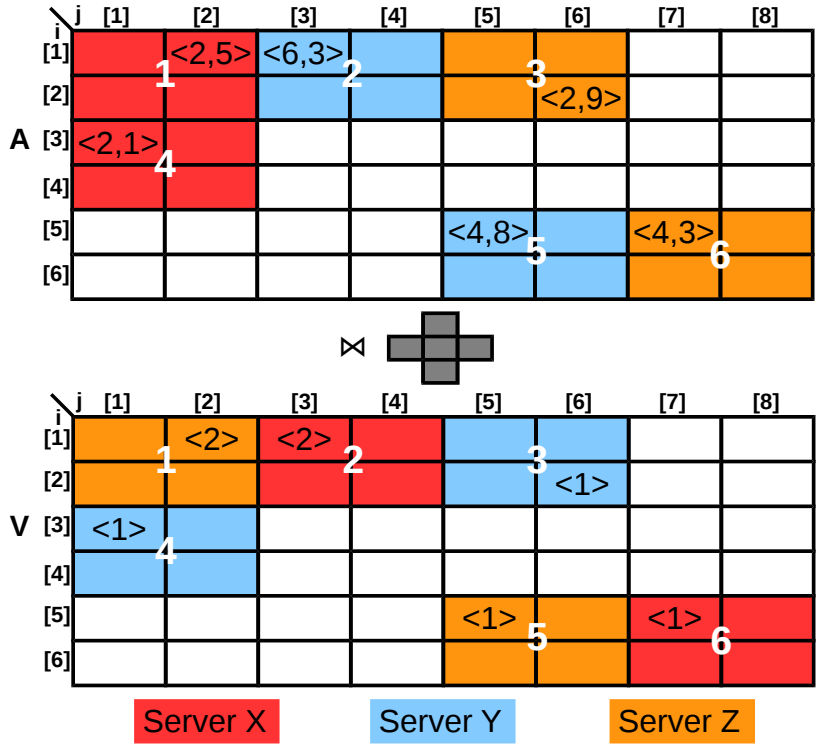
Contributions

- Formal array view definition
- Model incremental array view maintenance as an optimization formulation – including adaptive array and view reorganization
- Propose an effective three-stage heuristics
- Introduce an analytical cost model for query integration
- Experimentally compare against existing solutions on PTF catalog and LinkedGeoData

Incremental Array View Maintenance Axes

- Views over array similarity join (Zhao 2016)
  - Encompasses a larger variety of join predicates
  - Composable
- Batch updates – (He 2005, Katsis 2015)
  - Not only to reduce the per-transaction overhead
  - Share delta computation inside batch
- Update granularity – chunk (Brown 2010, Duggan 2014)
- Aggregates – standard SQL aggregate functions, e.g., SUM, COUNT, AVG (Yang 2001, Palpanas 2002)
- Distributed processing (Luo 2003, Liu 2005)
  - Potential join pairs for each chunk are magnified
- Recursive maintenance (Ahmad 2012, Nikolic 2016)

Array Views




```
CREATE ARRAY VIEW V AS
  SELECT COUNT(*) AS cnt
  FROM A A1 SIMILARITY JOIN A A2
       ON (A1.i = A2.i) AND (A1.j = A2.j)
       WITH SHAPE L1(1)
  GROUP BY A1.i, A1.j
```

Array View Definition

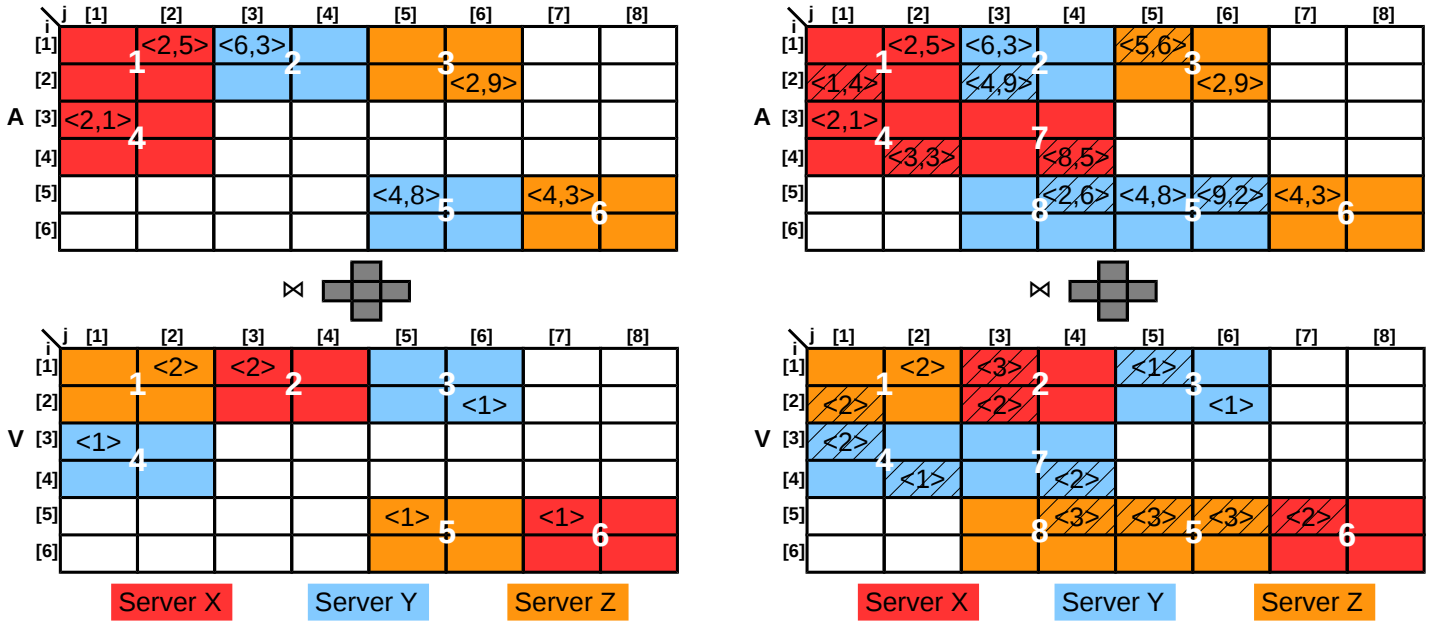
n multi-dimensional arrays $\alpha_1, \ldots, \alpha_n$
k unary array operators $\oplus_1, \ldots, \oplus_k$

$$V \leftarrow\!\shortmid \oplus_1 \left(\cdots \oplus_k \left(\alpha_1 \bowtie^{\mathcal{M}_1}_{\sigma_1, f_1} \alpha_2 \bowtie^{\mathcal{M}_2}_{\sigma_2, f_2} \cdots \bowtie^{\mathcal{M}_{n-1}}_{\sigma_{n-1}, f_{n-1}} \alpha_n \right) \cdots \right)$$

For example:

```
CREATE ARRAY VIEW V AS
  SELECT COUNT(*) AS cnt
  FROM A A1 SIMILARITY JOIN A A2
       ON (A1.i = A2.i) AND (A1.j = A2.j)
       WITH SHAPE L^1(1)
  GROUP BY A1.i, A1.j
```

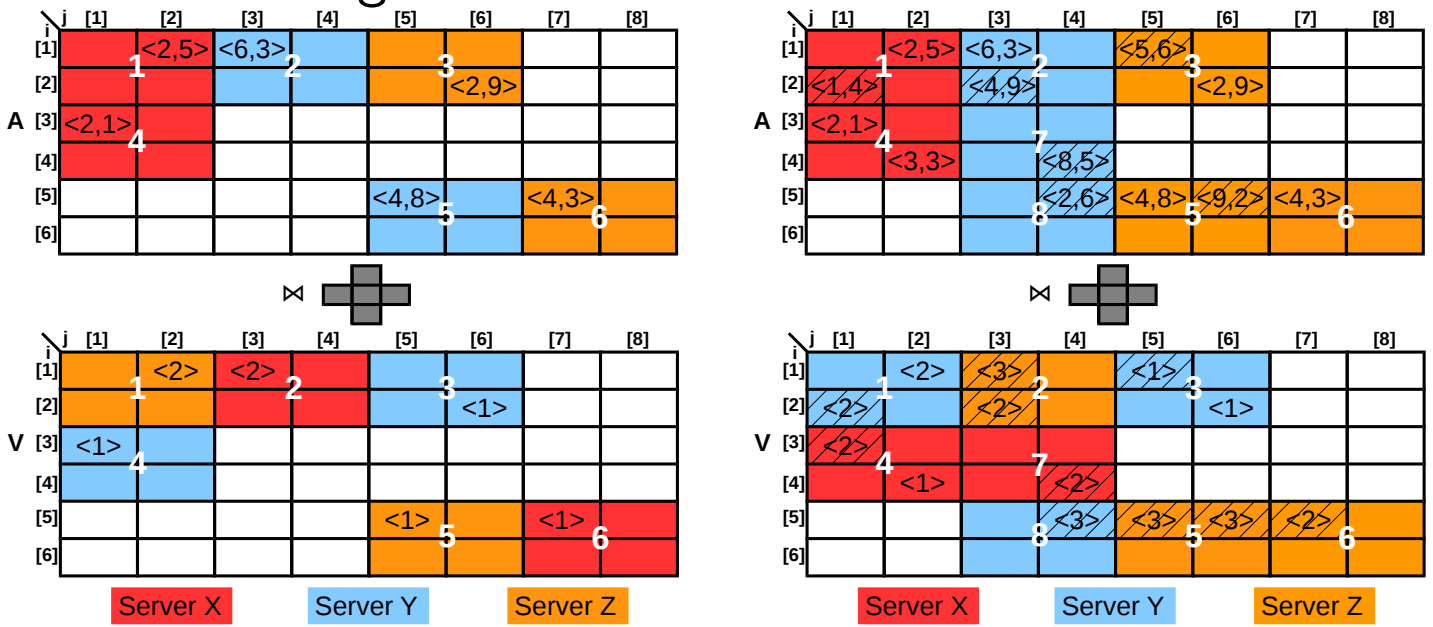
Baseline Array View Maintenance



- For each update batch ΔD:
  - Assign ΔD to node according to chunking strategy
  - $Q(D, \Delta D)$ – differential view computation
  - $M(D) + \Delta Q(D, \Delta D)$ – view aggregation
- Excessive communication
- Load imbalance

Optimal Array View Maintenance

- Array chunk (view chunk) reassignment – not stuck with an unfavorable static chunking strategy
- Incoming chunks are not first assigned to node based on pre-determined chunking strategy
- Piggyback on the chunk replication incurred by view maintenance when computing the reassignment
- A window of past batch updates is considered to avoid unstable reassignments



Analytical Cost Model

$$\min\left\{\lambda\cdot\max\left\{\max_k\left\{\sum_{i,j}^{k\neq j} x_{ikj}\cdot B_i T_{ntwk}+\sum_{p,q,v,j}^{k\neq j,(p,q,v)\in U_0} z_{pqk}\cdot y_{vj}\cdot B_{pq} T_{ntwk}\right\},\right.\right.$$
$$\left.\left.\max_k\left\{\sum_{p,q} z_{pqk}\cdot B_{pq} T_{cpu}\right\}\right\}$$
$$+(1-\lambda)\cdot\max\left\{\max_k\left\{\sum_{i,j}^{k\neq j} x'_{ikj}\cdot B_i T_{ntwk}+\sum_{p,q,v,j,l}^{k\neq j,(p,q,v)\in U_l} z'_{pqk}\cdot y_{vj}\cdot W_l B_{pq} T_{ntwk}\right\},\right.$$
$$\left.\left.\max_k\left\{\sum_{p,q,l}^{(p,q)\in U_l} z'_{pqk}\cdot W_l B_{pq} T_{cpu}\right\}\right\}\right\}$$

Network time to transfer chunks

Network time to transfer partial join results

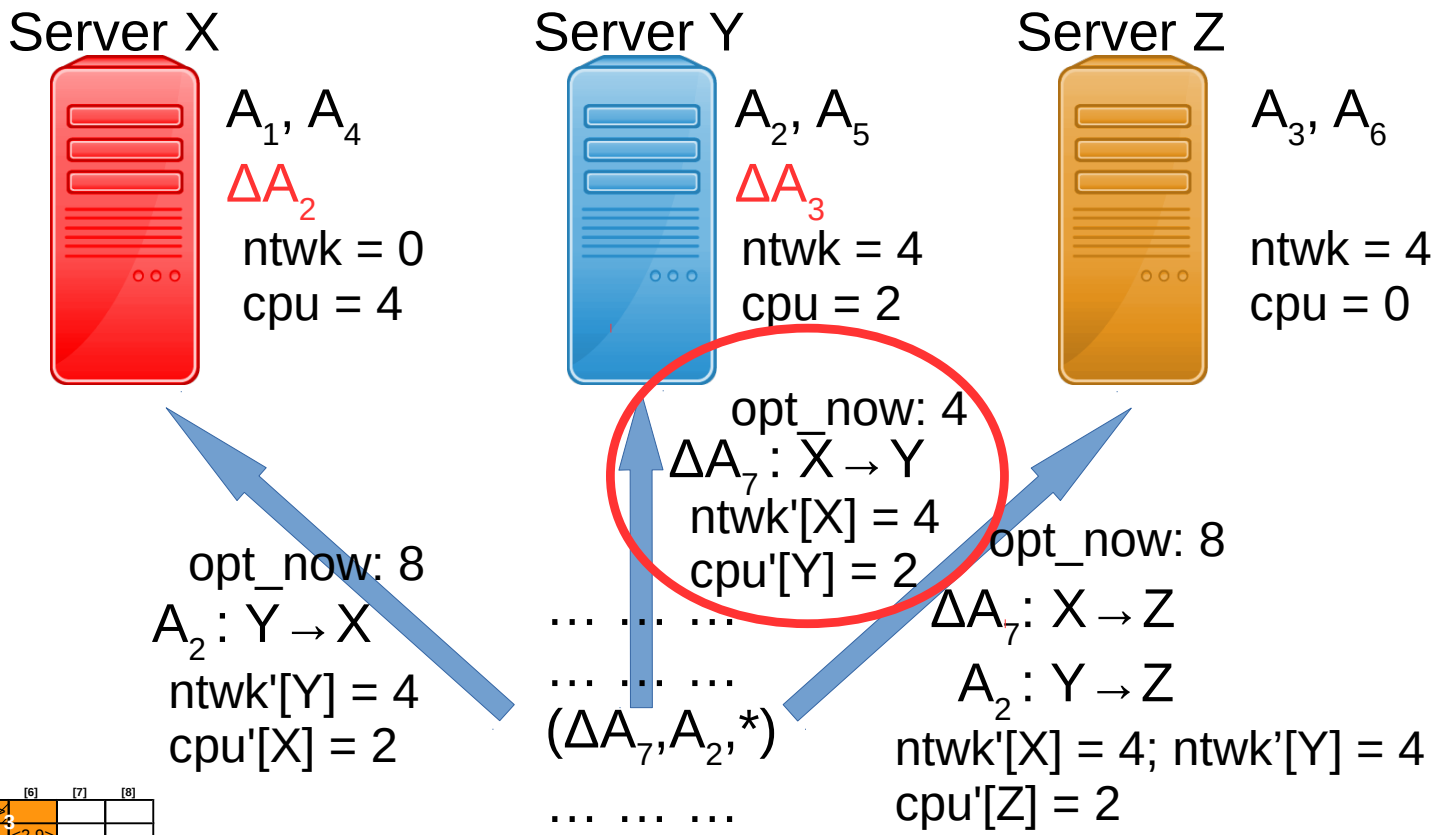
CPU time to compute aggregations

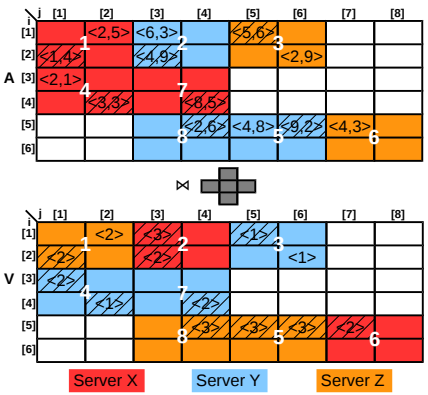
Three-stage Heuristic

Complicated formulation $\rightarrow$ decoupling

- Differential view computation $\rightarrow$ x_{ikj}, z_{pqk}
- View chunk reassignment $\rightarrow$ y_{vj}
- Array chunk reassignment $\rightarrow$ y_{aj}

Differential View Computation



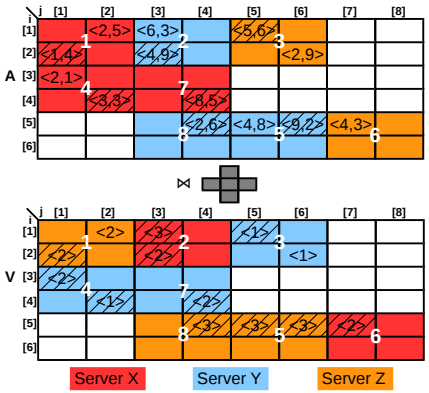


View Chunk Reassignment

server	*ntwk*	*cpu*
X	32	36
Y	36	30
Z	30	35

join result	server
$J_1 : \Delta A_1 \bowtie A_1$	X
$J_2 : \Delta A_4 \bowtie A_1$	X
$J_3 : \Delta A_2 \bowtie A_1$	Y

$V_1 \rightarrow$	transfers	*ntwk′*	*cpu′*	*opt_now*
X	J_3	$Y = 4$	$X = 6$	42
Y	J_1, J_2	$X = 8$	$Y = 6$	**40**
Z	J_1, J_2, J_3	$X = 8,\ Y = 4$	$Z = 6$	41

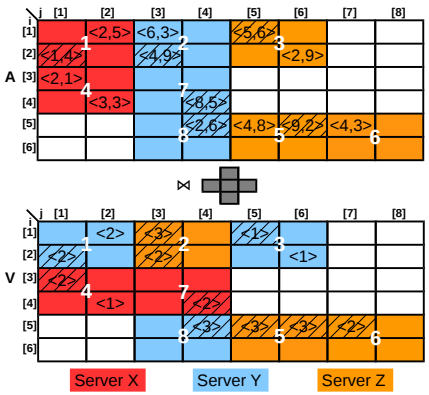


Array Chunk Reassignment

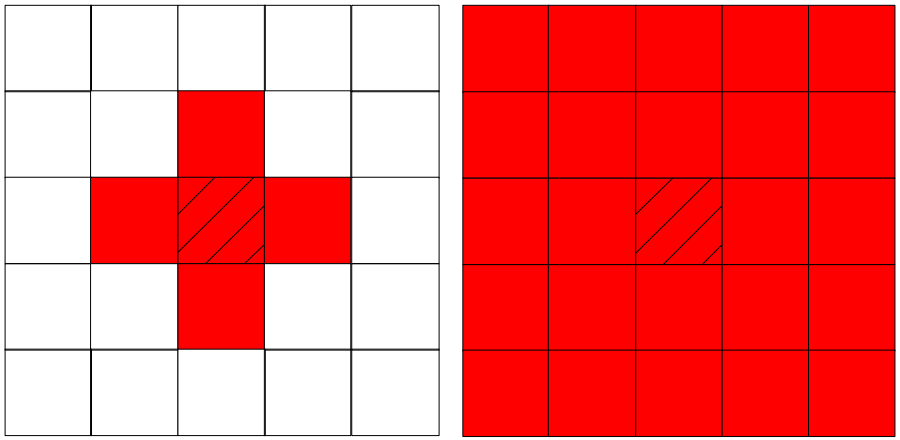
score	
A_2, V_1	8
A_1, V_1	6
A_1, V_2	4
A_2, V_3	4
A_3, V_3	2
...	

server	views	cpu_thr
X	V_2, V_6	4
Y	V_1, V_4, V_7	3
Z	V_3, V_5, V_8	1

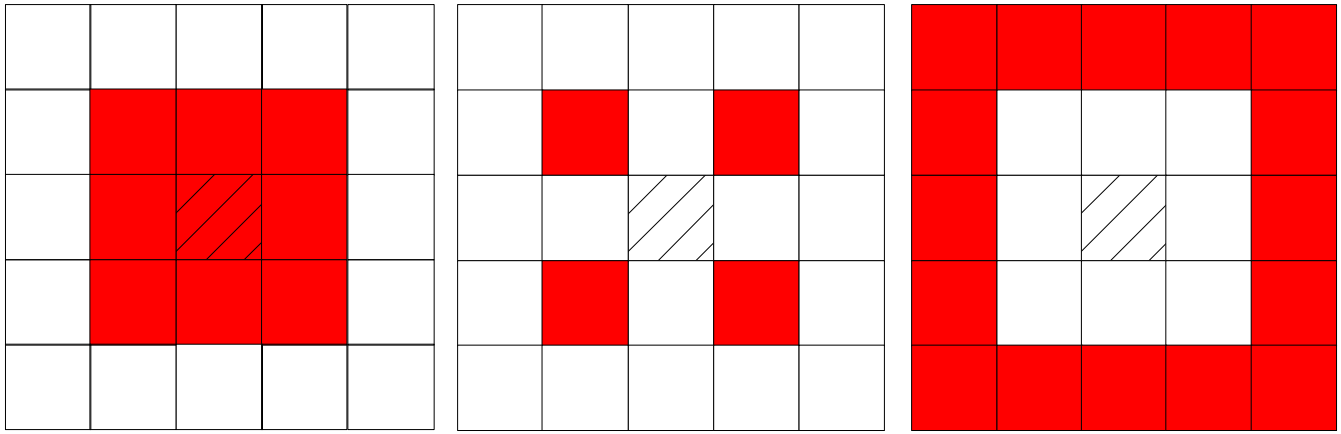
	A_1	A_2	A_3	...
size	1	1	1	...
replica	X, Z	Y, Z	Z, Y	



Query Integration



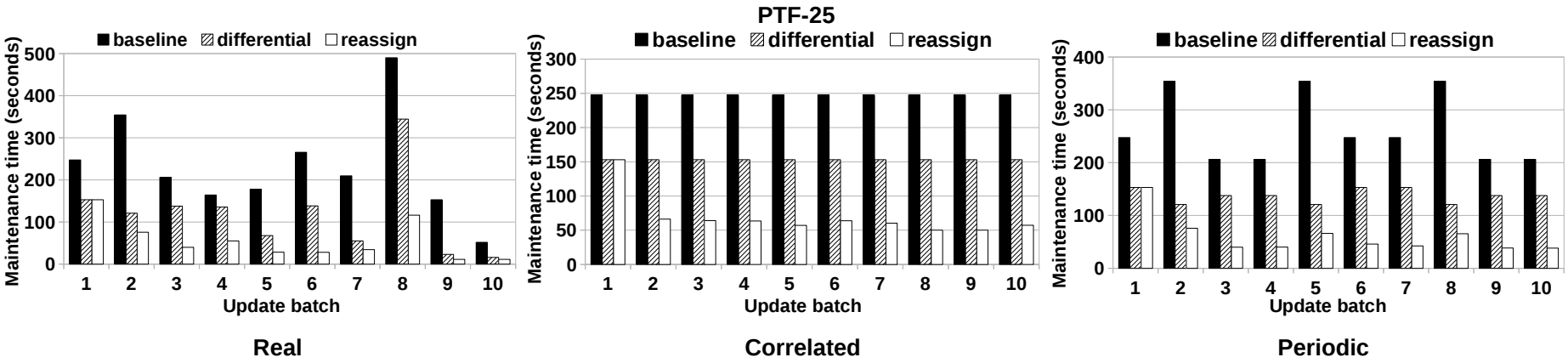
L1(1) L∞(2)



L∞(1) L∞(1) ← L1(1) L∞(1) ← L∞(2)

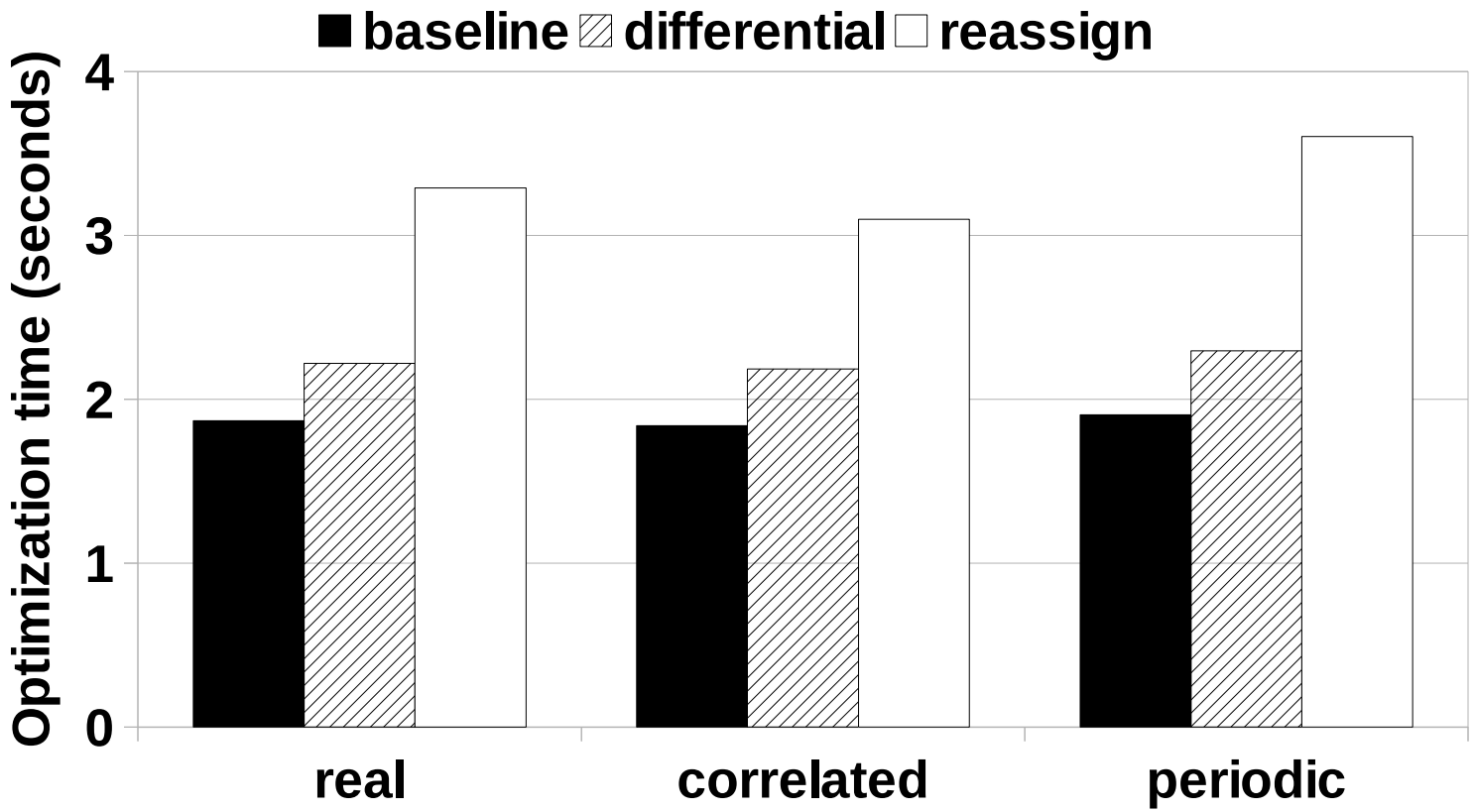
View Maintenance Time

`PTF[time=1,153064;ra=1,100000;dec=1,50000]`
1 billion time-stamped objects, 343 GB

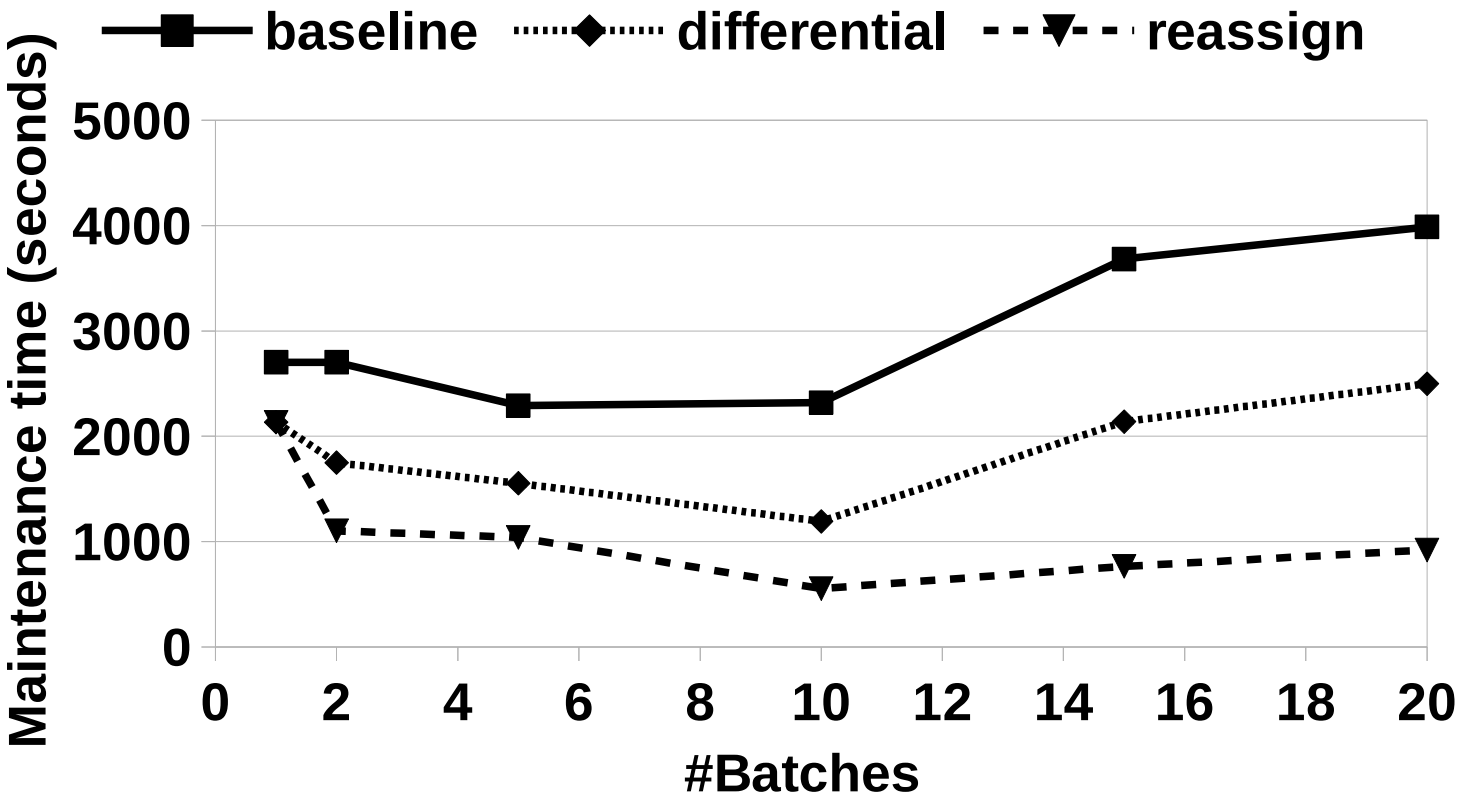


- Real: real updates from PTF pipeline
- Correlated: updates focusing on the same place
- Periodic: updates from several regions periodically – 1, 2, 3, 3, 2, 1, ...

Average Optimization Time per Update Batch



Sensitivity to #Batches



Recap

- Introduce materialized array views as a database construct for derived data products in science
- Model incremental array view maintenance with batch updates as an MIP optimization and give a three-stage heuristic that finds effective update plans
- Heuristic repartitions the array and the view continuously based on a window of past updates as a side-effect of view maintenance
- Design an analytical cost model for integrating materialized array views in queries
- Experimental results confirm the effectiveness of the heuristics and the quality of the maintenance plan
- Incrementally maintain the production PTF "association table" and help find rare supernovae at Berkeley Lab and CalTech

Outline

1. Array Similarity Join [SIGMOD 2016]
2. Array Views [SIGMOD 2017]
3. Distributed Caching of Raw Arrays [arXiv 2018]
4. ArrayUDF [HPDC 2017]
5. Collaborators

Raw PTF Data (FITS, HDF5, CSV)

A: Standard FITS Format

Figure FITS-1

B: Binary Table Extension

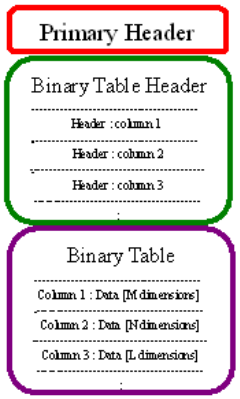


Figure FITS-2

```
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,
```

- A file for each night; file ranges overlap
- High object cardinality variance across files
- Files stored across a distributed cluster
- Load and partition inside array database before query

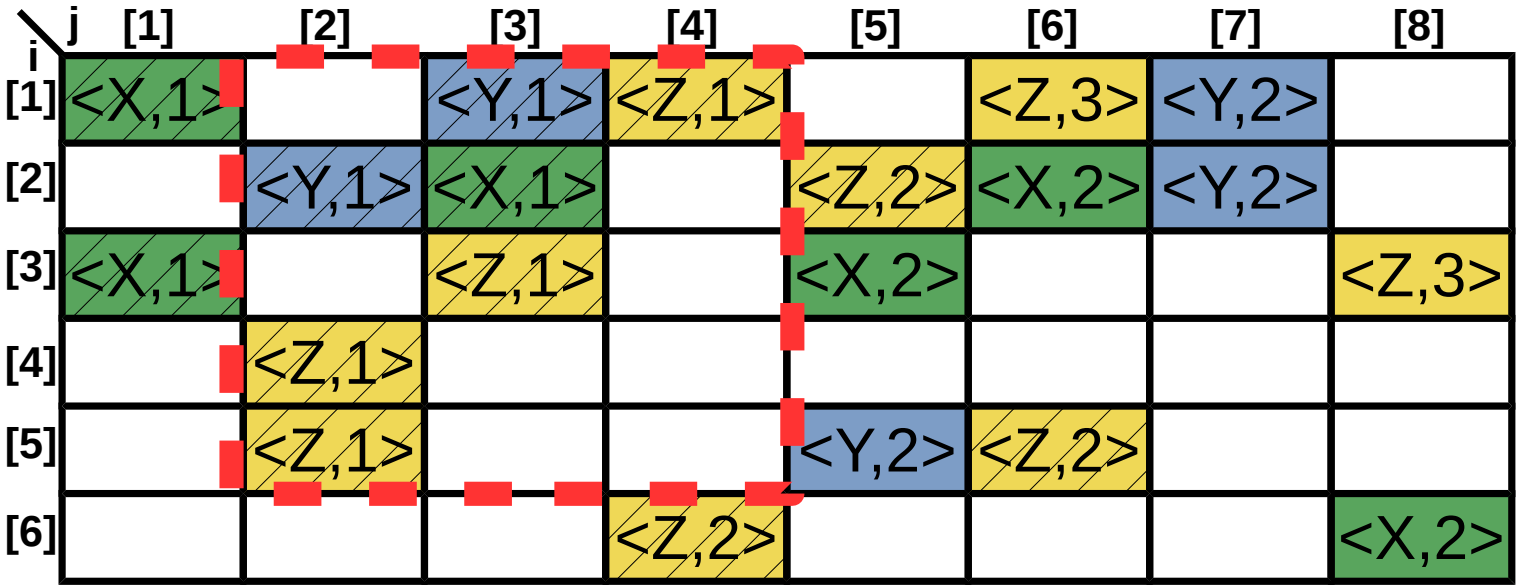
Contributions

- Design query-driven distributed caching framework for raw arrays
  - Identify cells to be cached locally and incrementally from each input file
  - Assign cells to nodes such that dependent cells are collocated
- Design an evolving R-tree index that refines the chunking of a sparse array to efficiently find the cells contained in a given subarray query
- Propose efficient cost-based algorithms for distributed cache eviction and placement that consider a historical query workload
- Evaluate experimentally distributed caching over raw arrays on two real sparse arrays with more than 1 billion cells stored in three file formats—FITS, HDF5, and CSV

Related Work

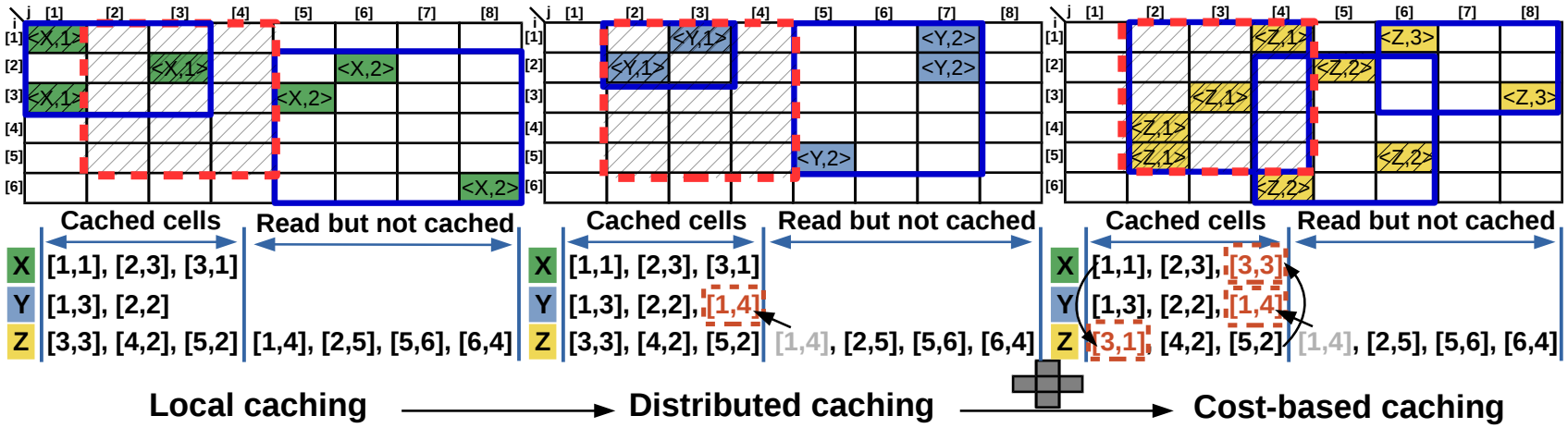
- Raw or in-situ data processing over CSV and JSON files
  - NoDB, SCANRAW, OLA-RAW, VIDa, SLALOM, Proteus
- Raw array processing in SciDB (HDF5)
  - ArrayBridge
- Caching of raw semi-structured data (JSON, Parquet)
  - NoDB, ReCache, vertical partitioning, invisible loading
- Distributed caching in relational databases and web systems
  - LRU-K, DBMIN, LRFU, LRU-Threshold, Lowest-Latency-First, and Greedy-Dual-Size
- Distributed caching in Hadoop and Spark

Raw Array Data Model



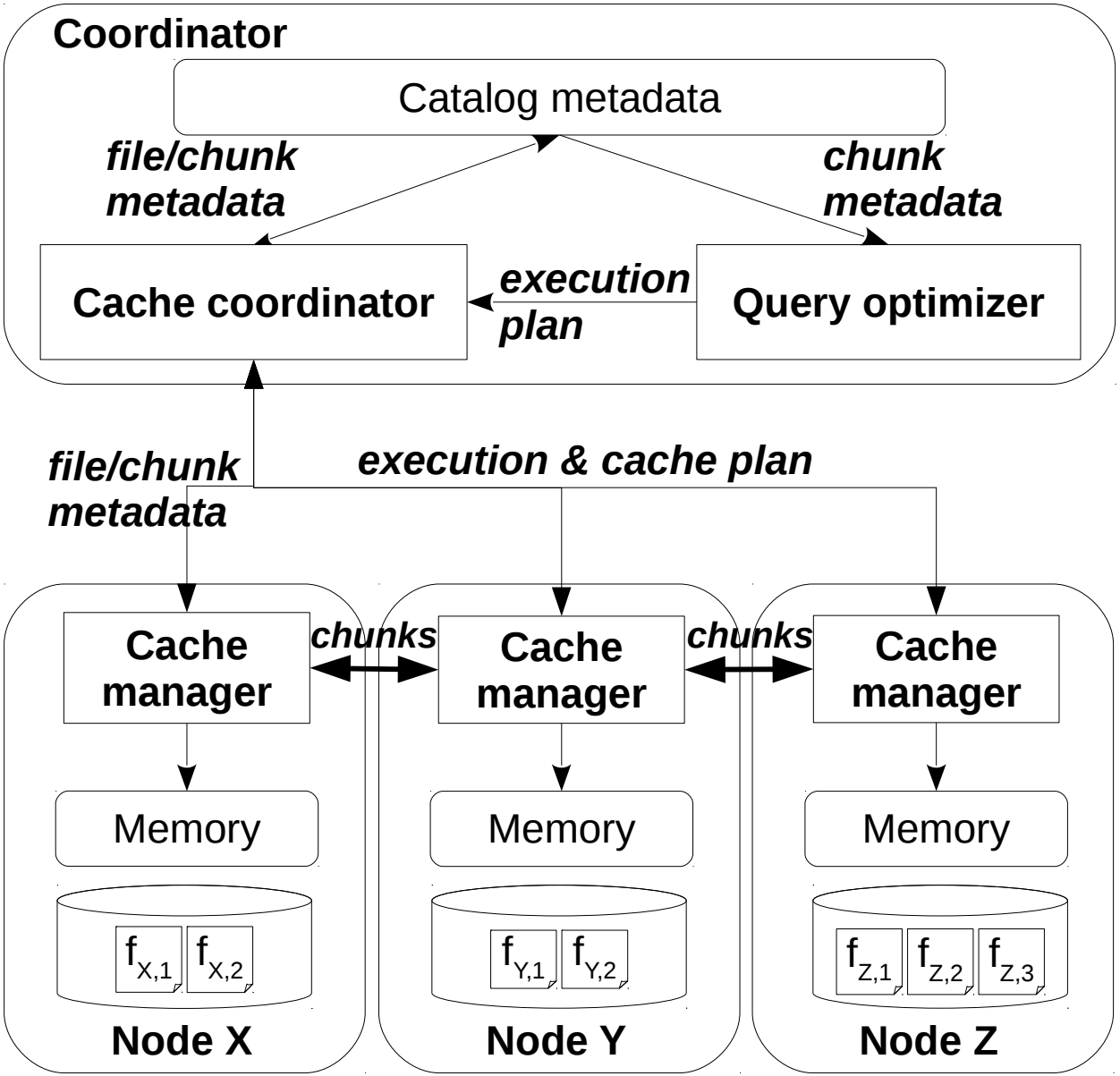
- Raw array `A<n:char,f:int>[i=1,6;j=1,8]` consisting of 7 files distributed over 3 nodes
- Attributes `n` and `f` correspond to the node and the file id on the node, e.g., `<X,2>` is a cell in the second file on node X
- The cell color also shows the server on which the file is stored
- The dashed rectangle specifies a query

Raw Array Distributed Caching Approach



- Build in-memory chunks incrementally based on the query (workload-driven chunking)
- Create higher granularity chunks for the entire file
- Design a global cost-based caching mechanism
- Combine query processing with cache replacement

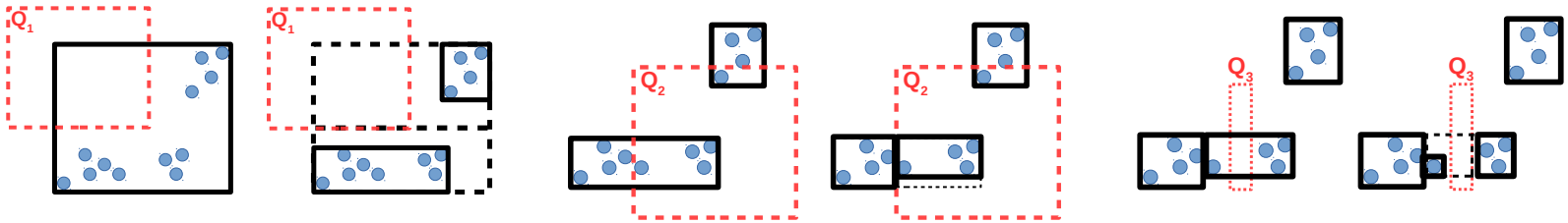
Distributed Caching Architecture



Raw Array Chunking

- Original raw array and resulting chunked array

- Query-driven chunking



Chunk Split Algorithm

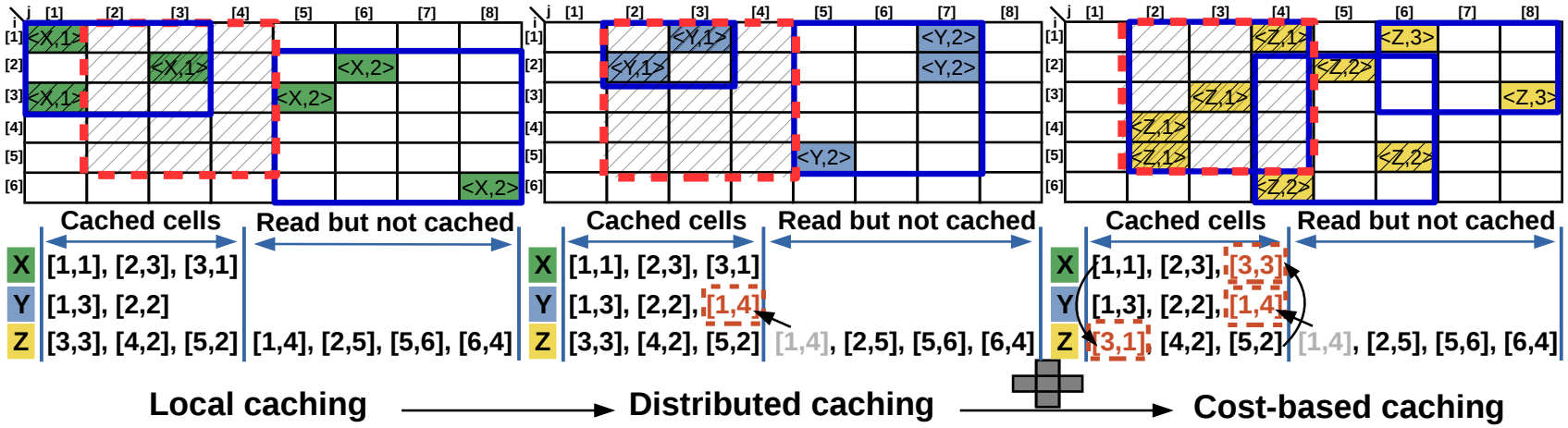
Input: Chunk α with bounding box $\mathcal{BB}_\alpha$ that intersects query subarray Q; Minimum number of cells threshold *MinC*

Output: Chunks β and γ after splitting α

1: **if**(cells in α ¡ *MinC*) **and** ($\exists$ cell in $\alpha \in Q$) **then return**
2: min_vol $= +\infty$
3: **for each** boundary $b \in Q$ that intersects with $\mathcal{BB}_\alpha$ **do**
4: $(\beta_b, \gamma_b) \leftarrow$ split cells in α into two sets by boundary b
5: **if** vol(β_b) + vol(γ_b) ¡ min_vol **then**
6: min_vol $\leftarrow$ vol(β_b) + vol(γ_b)
7: $\beta \leftarrow$ bounding_box(β_b), $\gamma \leftarrow$ bounding_box(γ_b)
8: **end if**
9: **end for**

- Build an evolving R-tree incrementally based on queries
- When to split? **Cell number threshold or no cells at all**
- How to split? **Always split into two chunks**

Cost-Based Caching



- Find optimal caching plan
  - Centralized setting: minimize cache misses
  - Distributed setting: minimize network misses
- Weigh chunks based on how they are accessed in the query workload
  - Access frequency: cache eviction
  - Co-locality: cache placement

Cache Eviction

- Scan a file entirely even if one accessed chunk is not cached
- $cost_{evict}(Q_l, f_i, \{C_j\}) = w_{Q_l} \cdot \frac{size(f_i)}{\sum size(uncached\ C_j)}$

Input: Cache state as set of triples $S = \{(Q_l, f_i, \{C_j\})\}$ consisting of the chunks j accessed from file i at query l; Set of pairs $(f_i, \{C_j\})$ consisting of the chunks j accessed from file i at the current query Q_{l+1}; Cumulated cache budget $B = \sum_k B_k$ consisting of local budgets B_k at each node

Output: Updated cache state $S' = \{(Q_{l+1}, f_i, \{C_j\})\}$

1: $S' \leftarrow \{(Q_{l+1}, f_i, \{C_j\})\}$
2: **while** $budget(S') < B$ **do**
3: Extract triple $t = (Q_l, f_i, \{C_j\})$ from S such that $budget(t \cup S') \leq B$ and $cost(t)$ is maximum
4: $S' \leftarrow S' \cup t$
5: $S \leftarrow S \setminus t$
6: Increase cost of triples $t' \in S$ that contain chunks in t
7: **end while**

Cache Placement

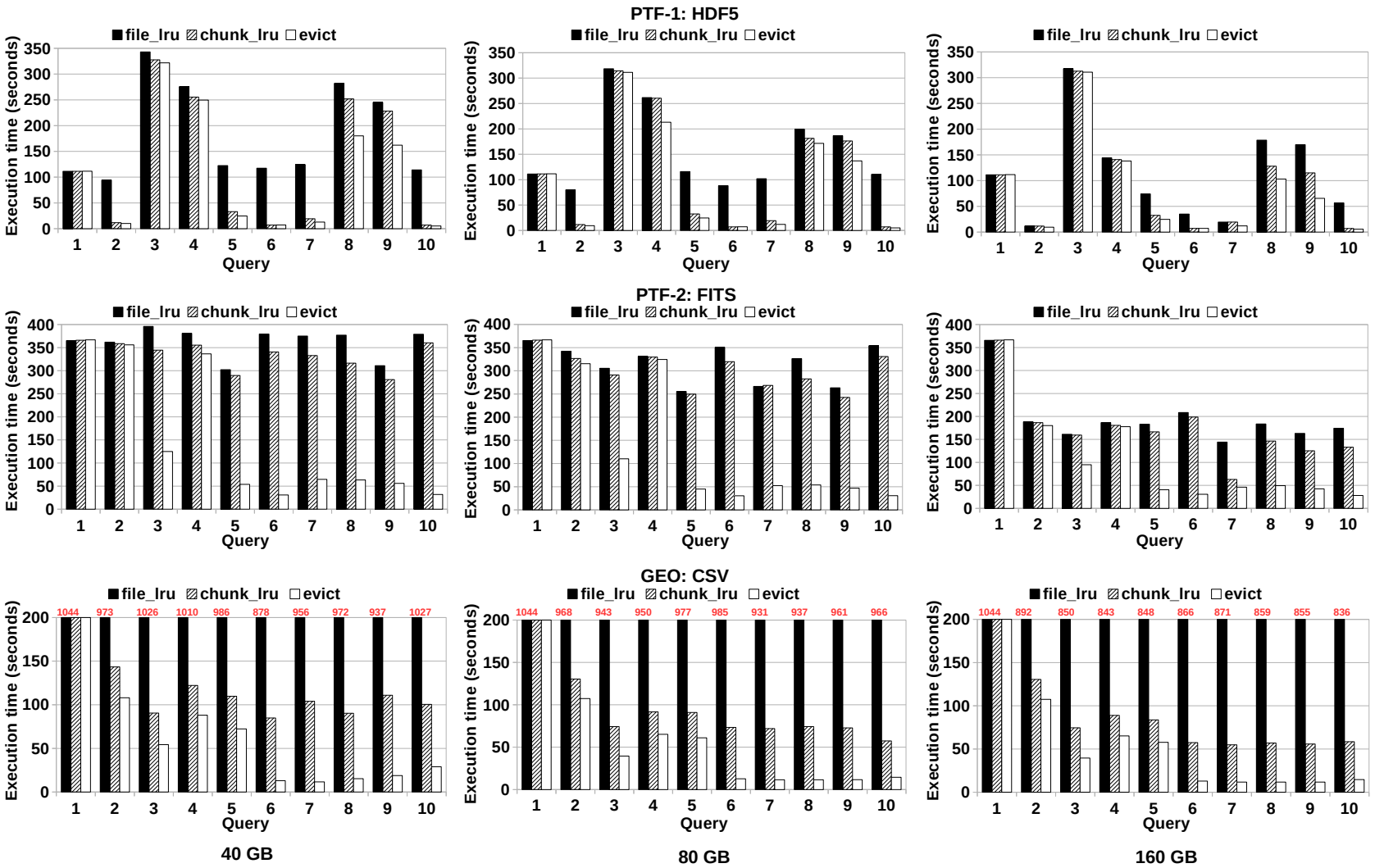
- Piggyback on the replication induced by query execution
- Preserve a single copy of every chunk
- Maximize co-locality of correlated chunks
- $cost_{place}(C_i, n, P', W) = \sum_{Q \in W} w_Q \cdot |C_j \in P'_n \wedge (C_i, C_j) \in Q|$

Input: Set $W = \{(Q_l, \{(C_i, C_j)\})\}$ consisting of chunk pairs (i, j) that join at query l; Set of locations $P = \{(C_i, \{N_k\})\}$ specifying all the nodes k that have a copy of cached chunk i at current query Q_{l+1}; Cache budget B_k at node k

Output: Updated locations $P' = \{(C_i, N_k)\}$

1: $P' \leftarrow \{p \in P\}$, where p has no replicas
2: **for each** $p = (C_i, \{N_k\}) \in P$ with multiple replicas **do**
3: Select node $n \in \{N_k\}$ such that $budget_n(C_i) \leq B_n$ and $cost(C_i, n, P', W)$ is maximum
4: $P' \leftarrow P' \cup (C_i, n)$
5: **end for**

Query Execution Time



Recap

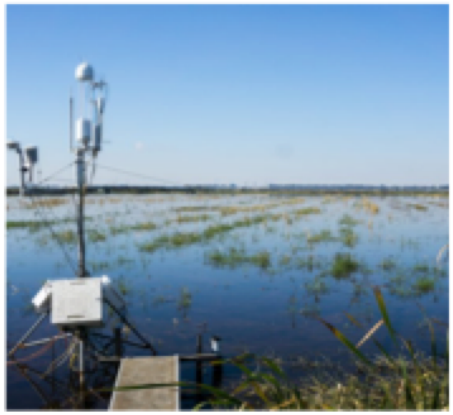
- Design query-driven distributed caching framework for raw arrays
- Design an evolving R-tree index that refines the chunking of a sparse array to efficiently find the cells contained in a given subarray query
- Propose efficient cost-based algorithms for distributed cache eviction and placement that consider a historical query workload

Outline

1. Array Similarity Join [SIGMOD 2016]
2. Array Views [SIGMOD 2017]
3. Distributed Caching of Raw Arrays [arXiv 2018]
4. ArrayUDF [HPDC 2017]
5. Collaborators

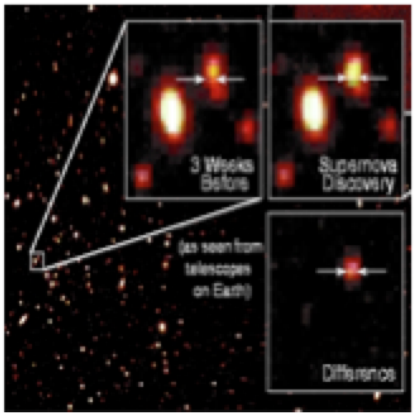
Scientific Data Analysis

1D Time Series



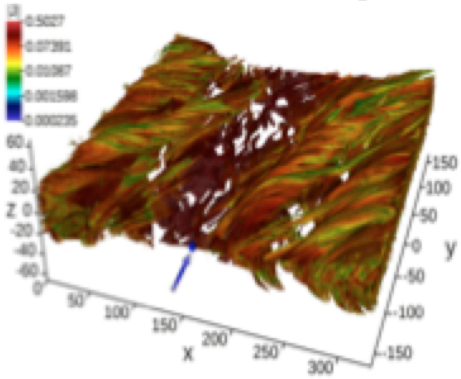
FLUXNET

2D Images



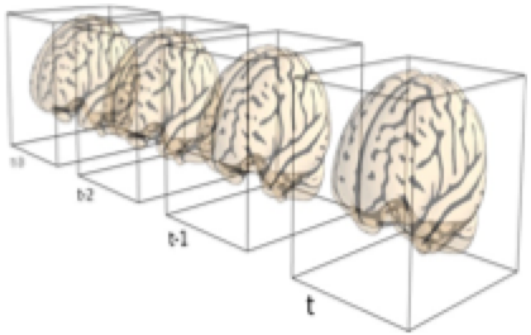
Supernovae Hunting

3D Meshed Space



Plasma Simulation

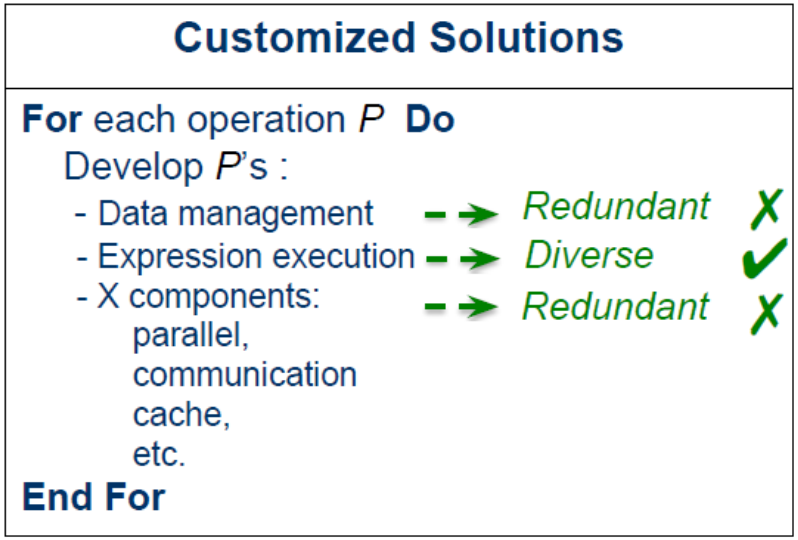
4D Functional Neuroimaging

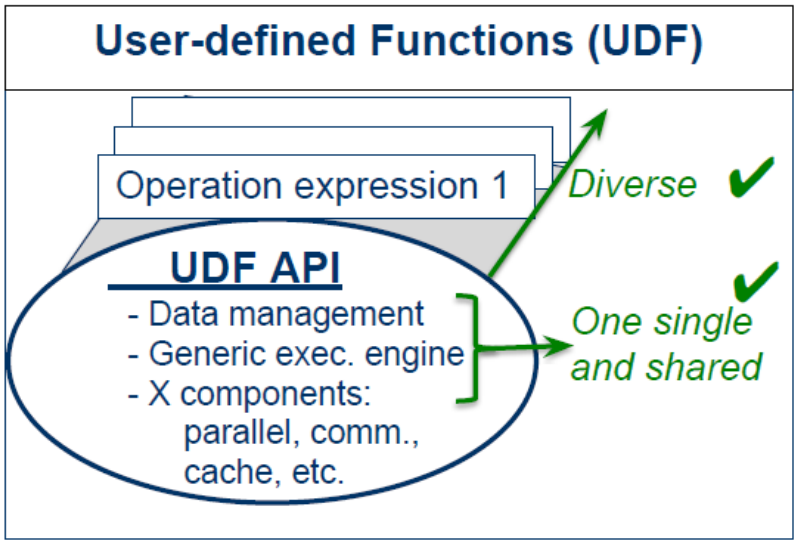


Metabolic Diseases Diagnose

- Process multi-dimensional arrays

Approaches for Scientific Data Analysis





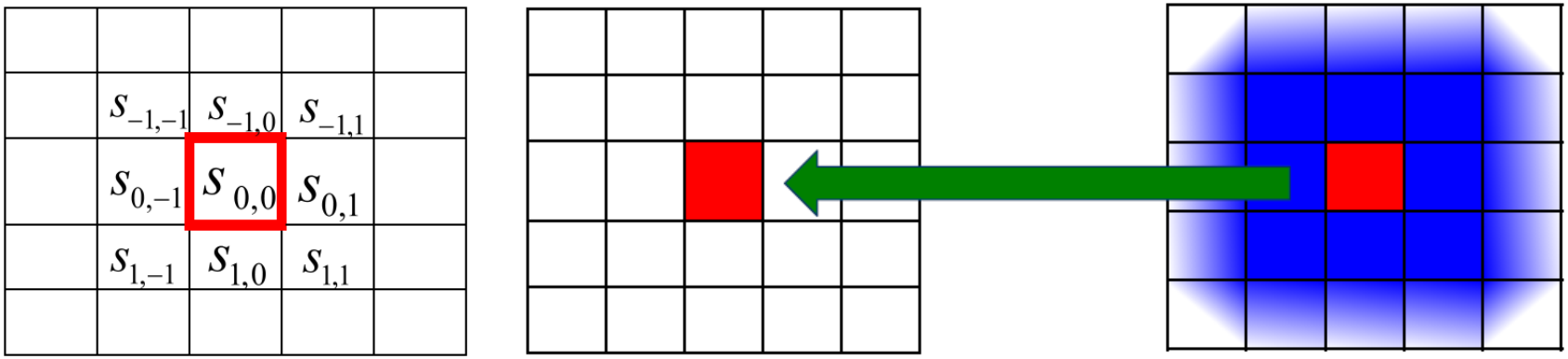
Amateurishly tuned ✗

Professionally tuned ✔

ArrayUDF Contributions

- Stencil-based User-Defined Functions (UDF)
  - Structural locality-aware array operations
- Native multi-dimensional array data model
  - In-situ processing on scientific data formats, e.g., HDF5
- Optimal chunking and ghost zone cells handling
  - Efficient parallel array processing on HPC systems

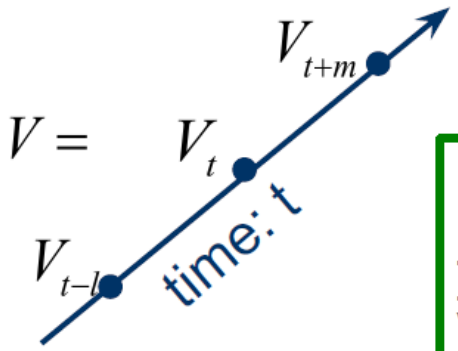
Stencil Computing Model



- d-dimensional stencil:
  $S = \{c_{i_1+\delta_1,\ldots,i_d+\delta_d} | \delta \in [L, R], L \in [-i, 0], R \in [0, N - i]\}$
- Materialized structural locality
- Flexible neighborhood expression
- Stencil UDF: $c'_{i_1,\ldots,i_d} = \mathcal{F}(\{S_{\delta_1,\ldots,\delta_d} | \delta \in [L, R]\})$

ArrayUDF Examples

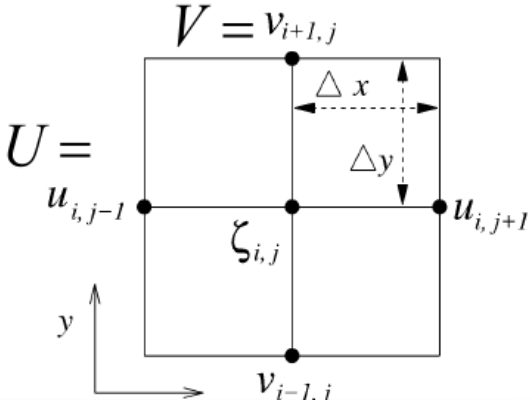
Example 1: Moving average



$$V'_t = \frac{w_t V_{t-k} + \ldots + w_t V_t + \ldots + w_{t+m} V_{t+m}}{k+m+1}$$

```
UDF_MV(Stencil s){
   return (s(-k)*w_{t-k}+ … s(0)*w_t+ … s(m)*w_{t+m})/(k+m-1)
}
V.Apply(UDF_MV)
```

Example 2: Vorticity computation

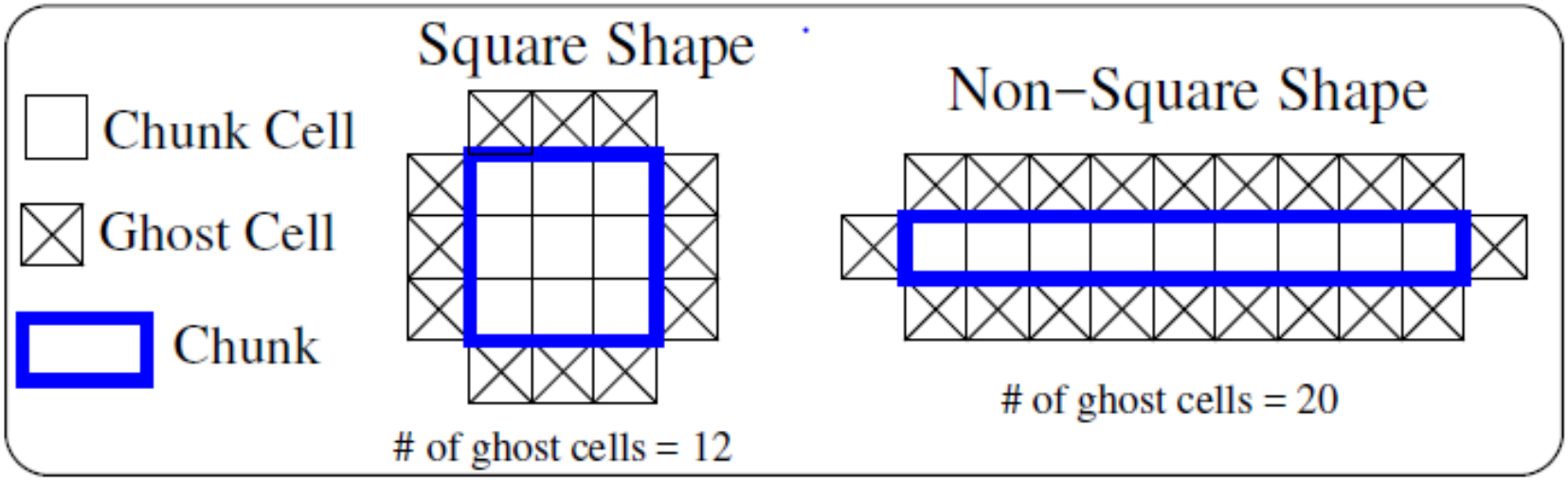


$$\xi_{i,j} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta x} + \frac{v_{i+1,j} - v_{i-1,j}}{2\Delta y}$$

```
UDF_VC_U(Stencil u){
   return u(0,1)- u(0, -1)
}
U.Apply(UDF_VC_1)
```

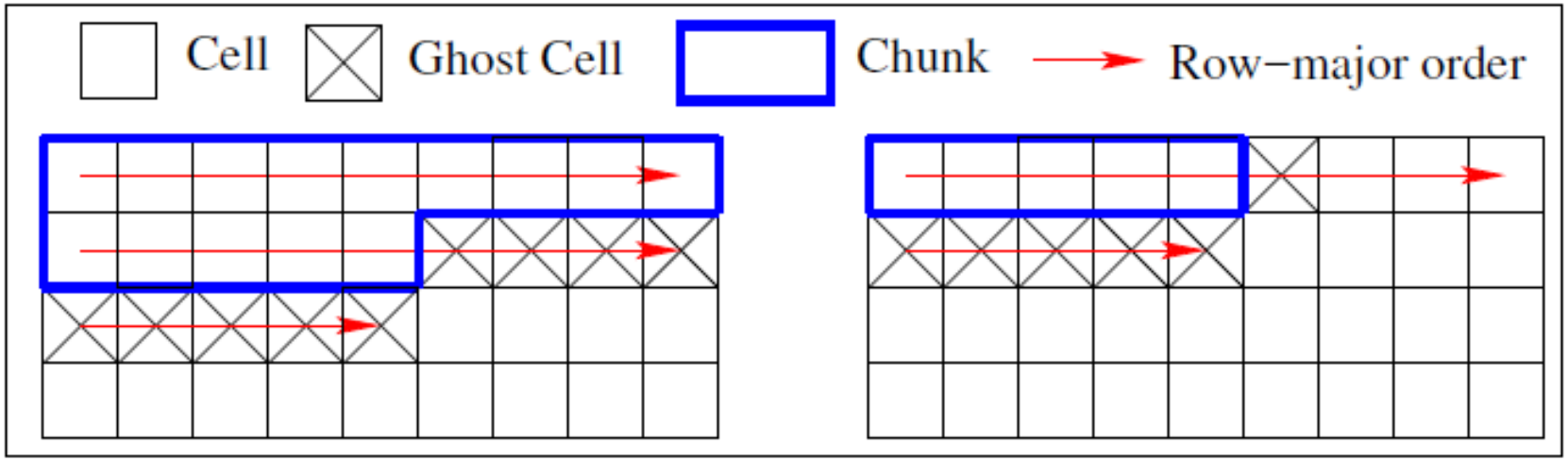
```
UDF_VC_V(Stencil v){
   return v(1,0)- u(-1, 0)
}
V.Apply(UDF_VC_V)
```

Chunking



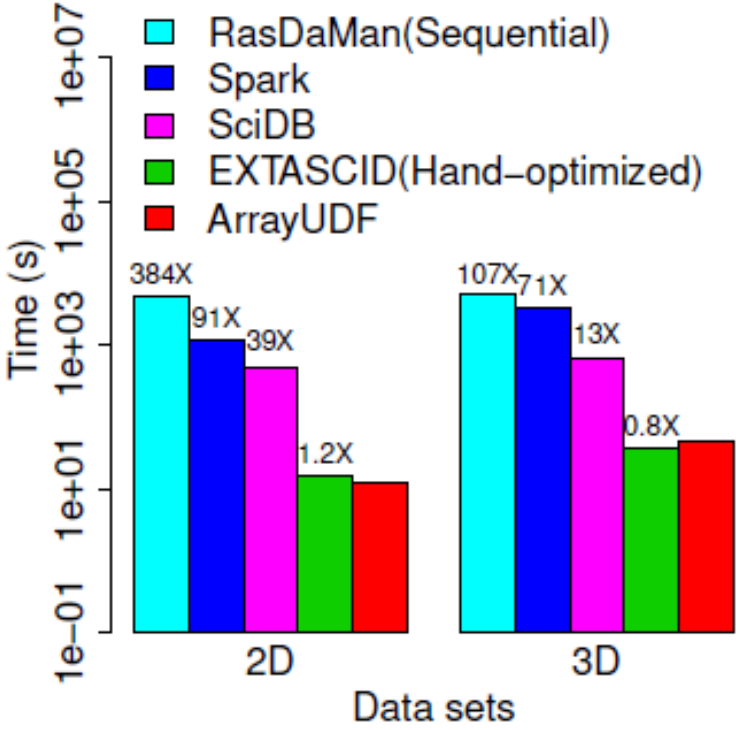
- Chunking enables parallel and out-of-core processing
- General chunking
  - Layout unknown
  - Minimize number of ghost cells
- Layout-aware chunking
  - Row-major
  - Maximize contiguous read from storage, e.g., disk

Ghost Zone

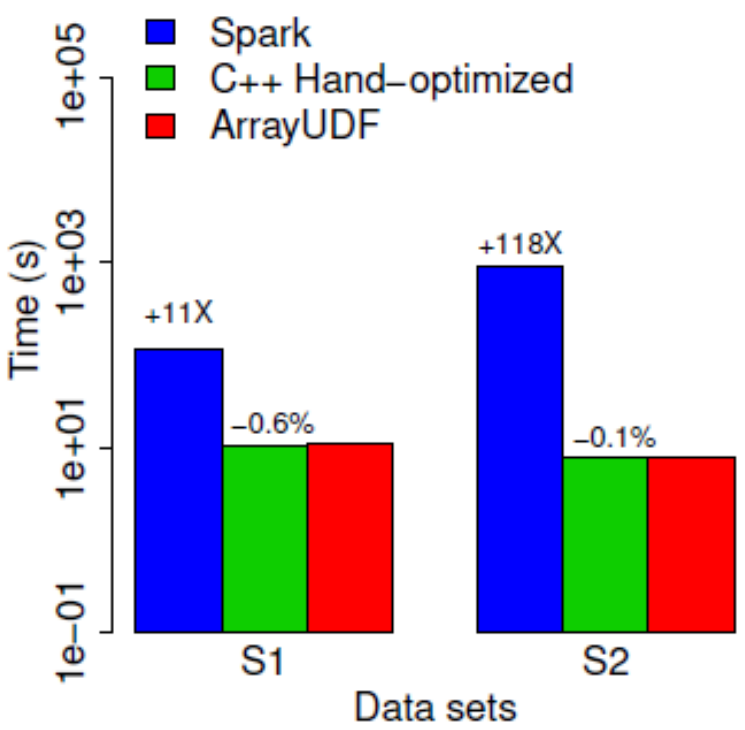


- ArrayUDF processes chunks in parallel and out-of-core
  - Ghost zone avoids communications between chunks
- Use trial-run to find ghost zone size
  - Run UDF on a sample Stencil instance that collects the offsets applied within UDF

Experiments



(a) Time for evaluating "window" operators.



(b) Time for solving the Poisson equation solver.

- ArrayUDF has close performance to hand-optimized code
- ArrayUDF is at least an order of magnitude faster

Recap

- ArrayUDF: User-Defined Scientific Data Analysis on Arrays
- Stencil-based UDF for structural locality-aware operations
- Native array model & in-situ processing on HDF5 files, etc.
- Optimal chunking and ghost zone methods for large arrays
- Provides performance close to hand-optimized code
- Is orders of magnitude faster than other systems
- Source code: `https://bitbucket.org/arrayudf/`

Outline

1. Array Similarity Join [SIGMOD 2016]
2. Array Views [SIGMOD 2017]
3. Distributed Caching of Raw Arrays [arXiv 2018]
4. ArrayUDF [HPDC 2017]
5. Collaborators

Publications and Collaborators

- Weijie Zhao, Florin Rusu, Bin Dong, and Kesheng Wu. Similarity Join over Array Data. In *SIGMOD 2016*.
- Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu, and Peter Nugent. Incremental View Maintenance over Array Data. In *SIGMOD 2017*.
- ..., Anna Y. Q. Ho, ..., Peter Nugent, ..., Florin Rusu, ..., Weijie Zhao. Illuminating Gravitational Waves: A Concordant Picture of Photons from a Neutron Star Merger. In *Science 2017*.
- Anna Y.Q. Ho, S. R. Kulkarni, Peter Nugent, Weijie Zhao, Florin Rusu et al. iPTF Archival Search for Fast Optical Transients. In *Astrophysical Journal Letters 2018*.
- Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu, Anna Y. Q. Ho, and Peter Nugent. Distributed Caching for Complex Querying of Raw Arrays. In *arXiv 2018*.

Thank you!

Questions?