Optimizing Parallel Algorithms for All Pairs Similarity Search

Maha Alabduljalil, Xun Tang, Tao Yang

Department of Computer Science

University of California at Santa Barbara

UCSB

All Pairs Similarity Search (APSS)

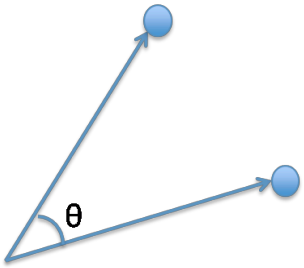
- **Finding all pairs of objects with similarity score above a threshold.**
- **Example Applications:**
  - Collaborative filtering/recommendation.
  - Coalition detection for advisement frauds.
  - Query suggestions.
  - Spam and near duplicate detection.

- **Slow data processing for large datasets**

Problem Definition

- **Cosine-based similarity:**

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



- **Given *n* normalized vectors, compute all pairs of vectors such that**

$$Sim(d_i, d_j) = \cos(d_i, d_j) = \sum_{t \in (d_i \cap d_j)} w_{i,t} \times w_{j,t} > \tau$$

- Quadratic complexity O(n^2)

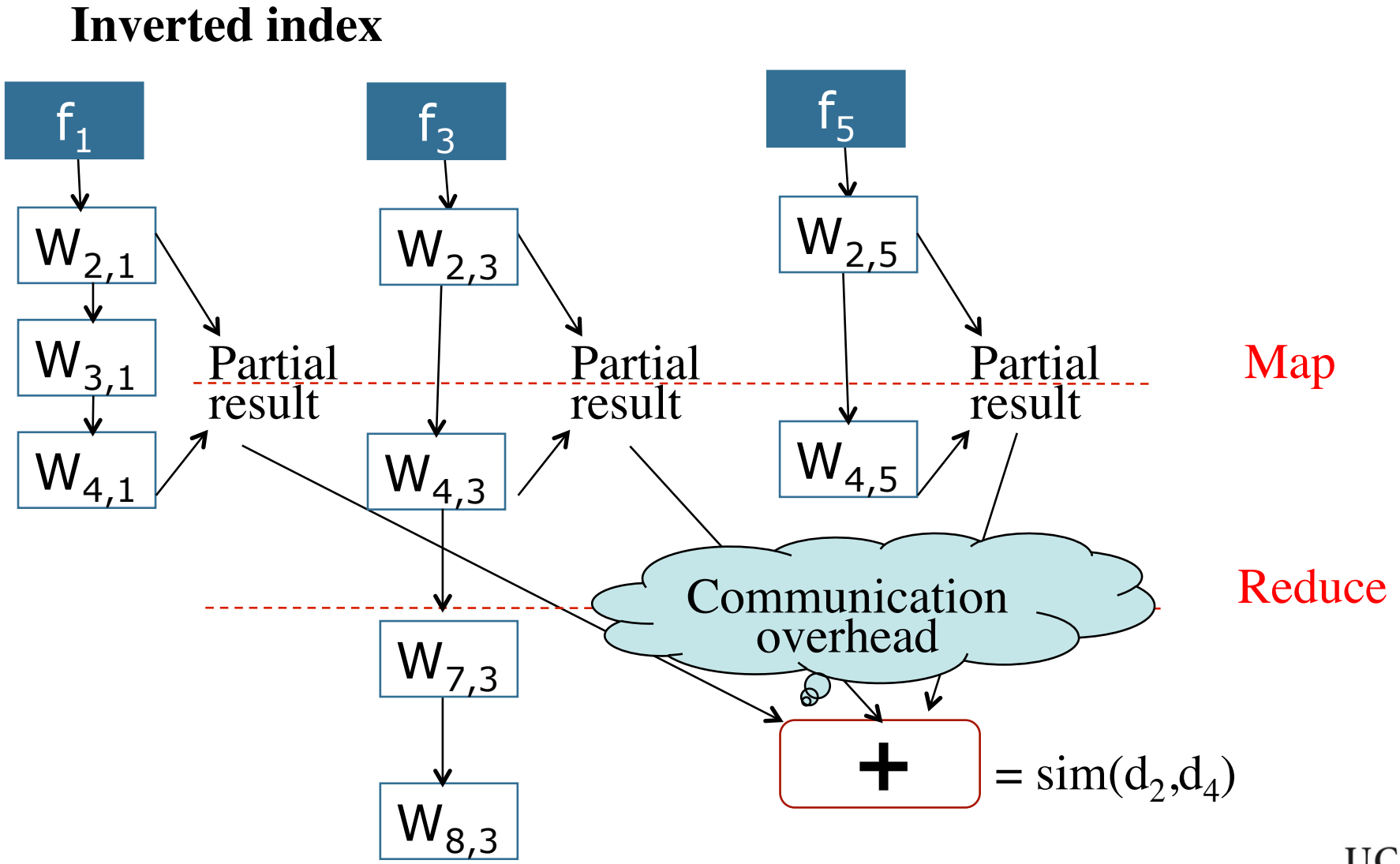
UCSB

Previous Work to Speedup Similarity Search

- **Approximated clustering**
  - LSH mapping [Gionis et al. VLDB99]: Tradeoff in precision, recall, & redundant comparison.
    - Proposed techniques can be used with LSH.
- **Exact similarity search:**
  - Dynamic computation filtering [Bayado et al. WWW07]
  - Inverted indexing to search vectors that share features[Arasu et al. VLDB06]
  - Parallel algorithms with MapReduce [Lin SIGIR09, Baraglia et al. ICDM10] Exploit parallelism in partial similarity score accumulation.

UCSB

Parallelism in Partial Similarity Score Accumulation



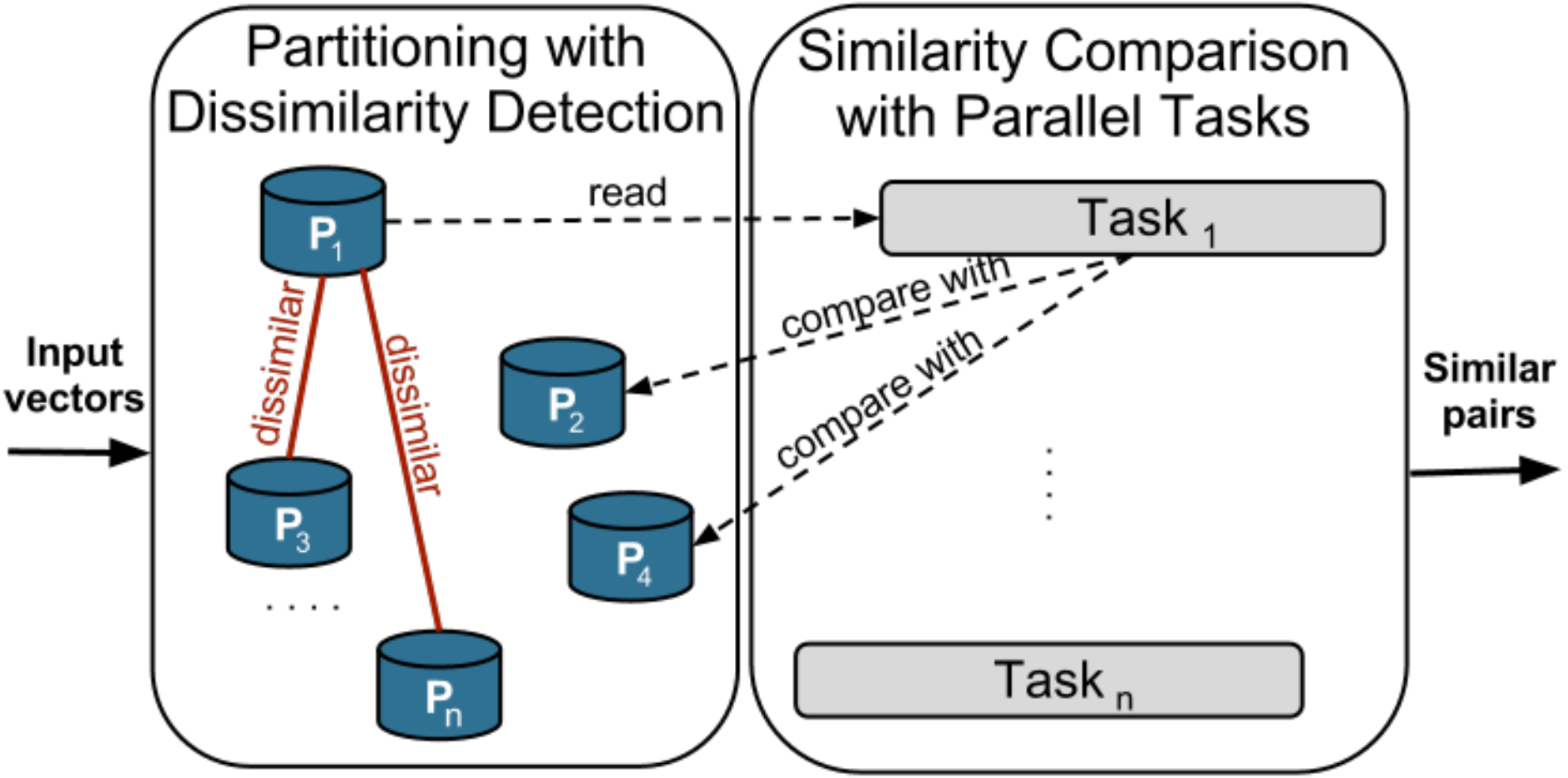
UCSB

Focus and contribution: Partition-based APSS

- **Focus on exact APSS with cosine similarity.**
- **New techniques to speedup APSS:**
  - Static partitioning to identify dissimilar vectors in advance.
    - → Early removal of I/O, communication, computation.
  - Partition-based symmetric comparison and load balancing.
    - → Removes unnecessary I/O and communication before computation.
  - No reduce tasks to avoid partial result parallelism.
    - → removal of excessive communication.
- **An order of magnitude of performance improvement.**

UCSB

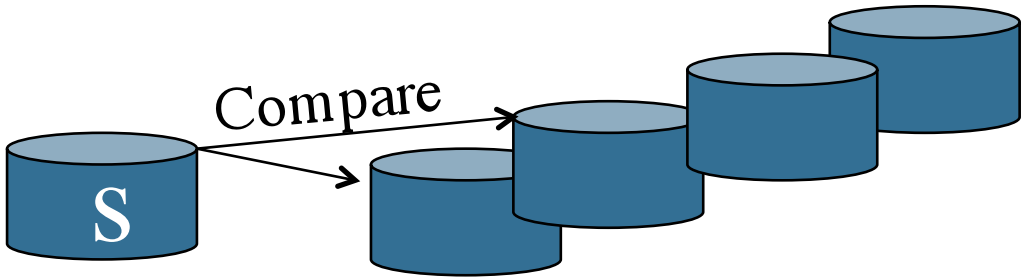
Partition-based Similarity Search (PSS)



UCSB

Function of each PSS task

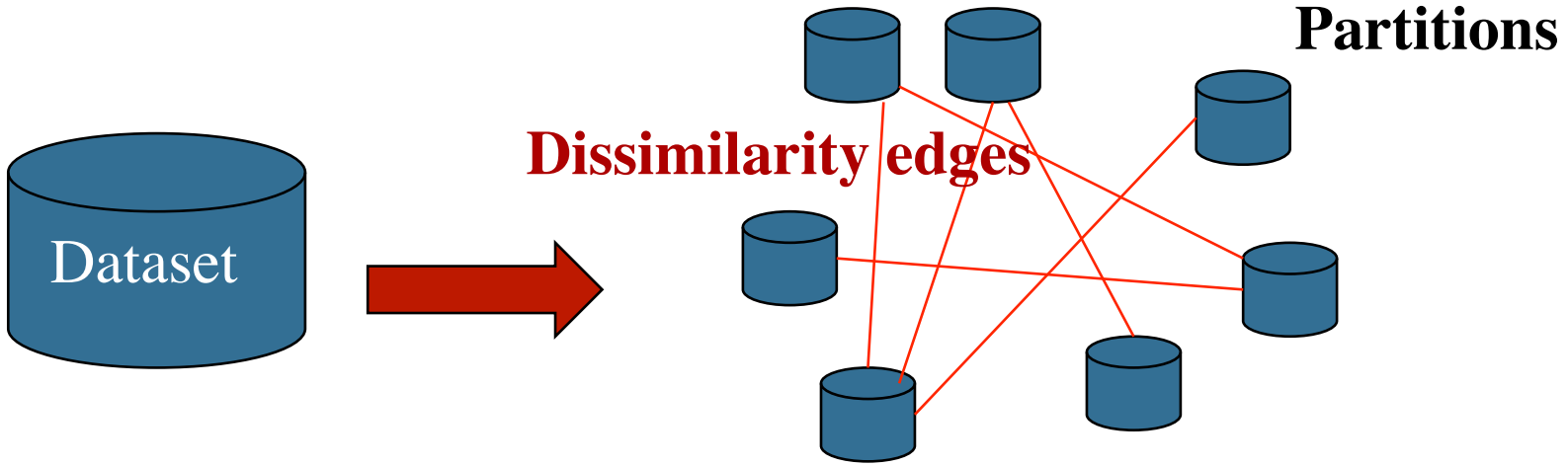
- Read assigned partition and build inverted index in memory area S.
- **Repeat**
  - Read vectors from other partitions
  - Compare S with these vectors
  - Write similar vector pairs
- **Until** other potentially similar vectors are compared.



Coarse-grain task parallelism

I. Static Partitioning with Dissimilarity Detection

- **Goal:**
  - Place dissimilar vectors into different partitions.
  - Low-cost partitioning:
    - near linear complexity in identifying pair-wise dissimilarity with no false positive.



How To Detect Dissimilar Vectors?

- **For normalized cosine similarity, Holder inequality:**

$$Sim(d_i, d_j) = \sum_{t \in (d_i \cap d_j)} w_{d_i,t} \times w_{d_j,t}$$

$$\leq min(maxw(d_i) * ||d_j||_1, maxw(d_j) * ||d_i||_1) \leq \tau$$

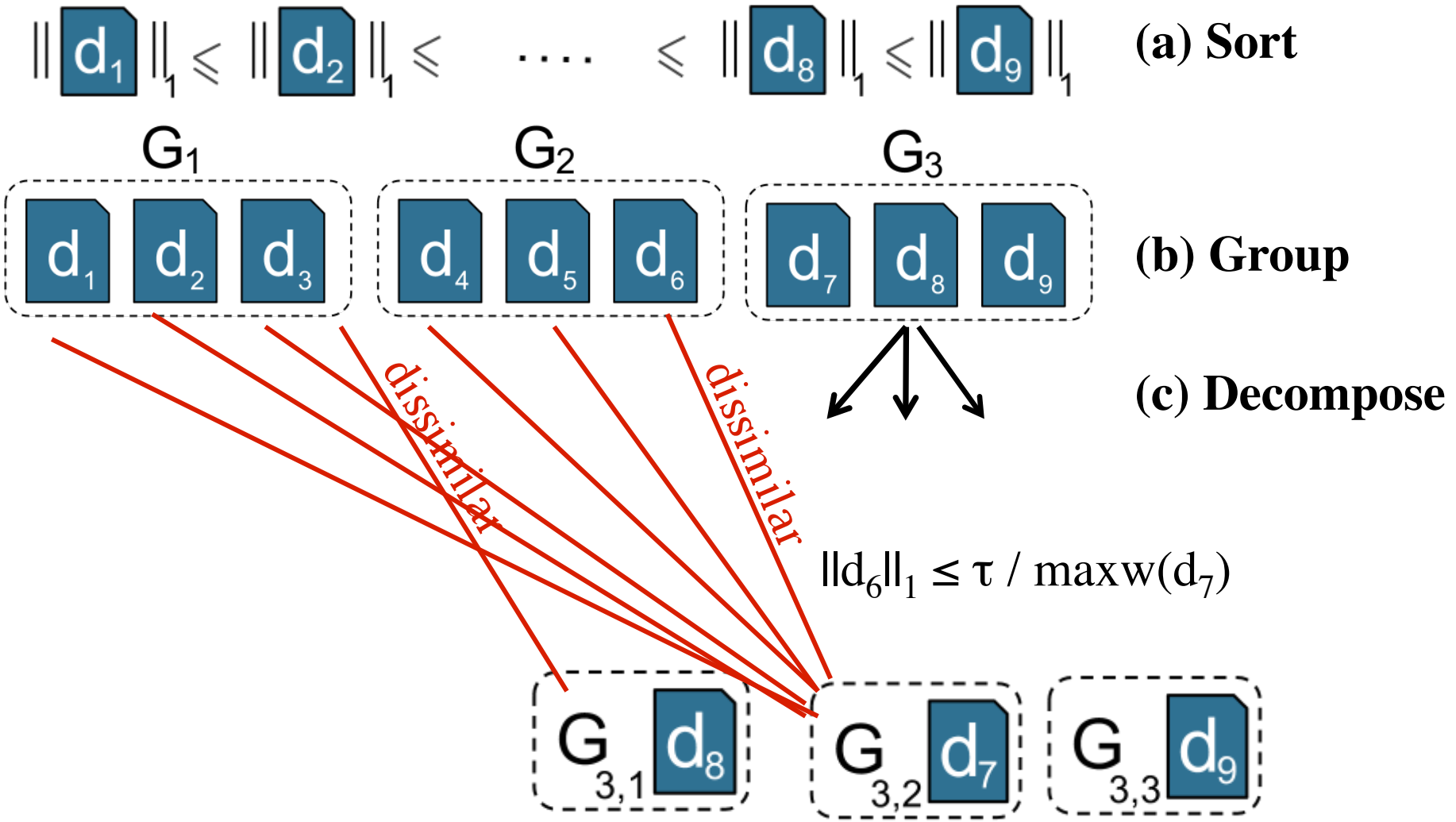
- d_i is dissimilar to d_j if:

$$\|d_i\|_1 \leq \frac{\tau}{maxw(d_j)}$$

- **How to detect dissimilarity without pair-wise comparison?**

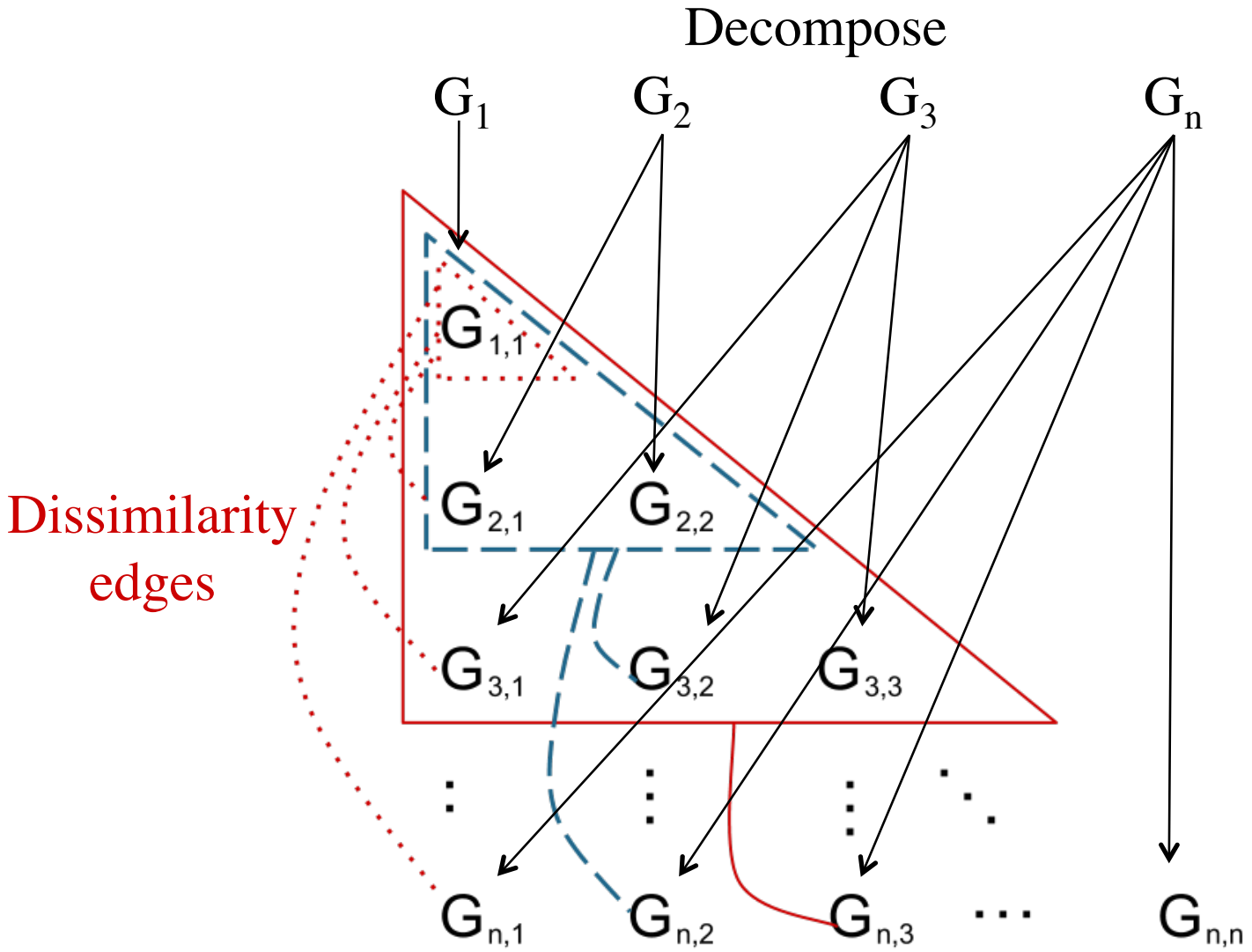
UCSB

Illustration of O(n*log*n) static partitioning with dissimilarity detection



UCSB

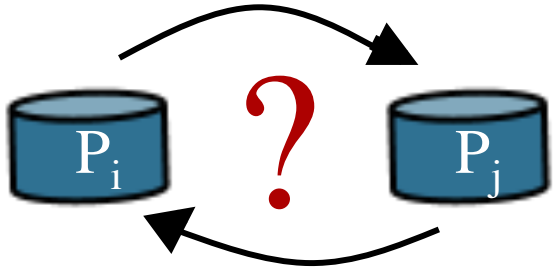
Final Output of Static Partitioning



II. Partition-oriented symmetric comparison

- **Issue:**
  - $Sim(d_i, d_j)=Sim(d_j, d_i)$.
    - →Redundant computations.
      - →Use vector IDs to detect symmetry ie. use one pair.
        - →Eliminate computation, but not I/O.

- **Strategy**
  - Partition-level symmetric detection
    - Eliminate entire partition I/O and communication

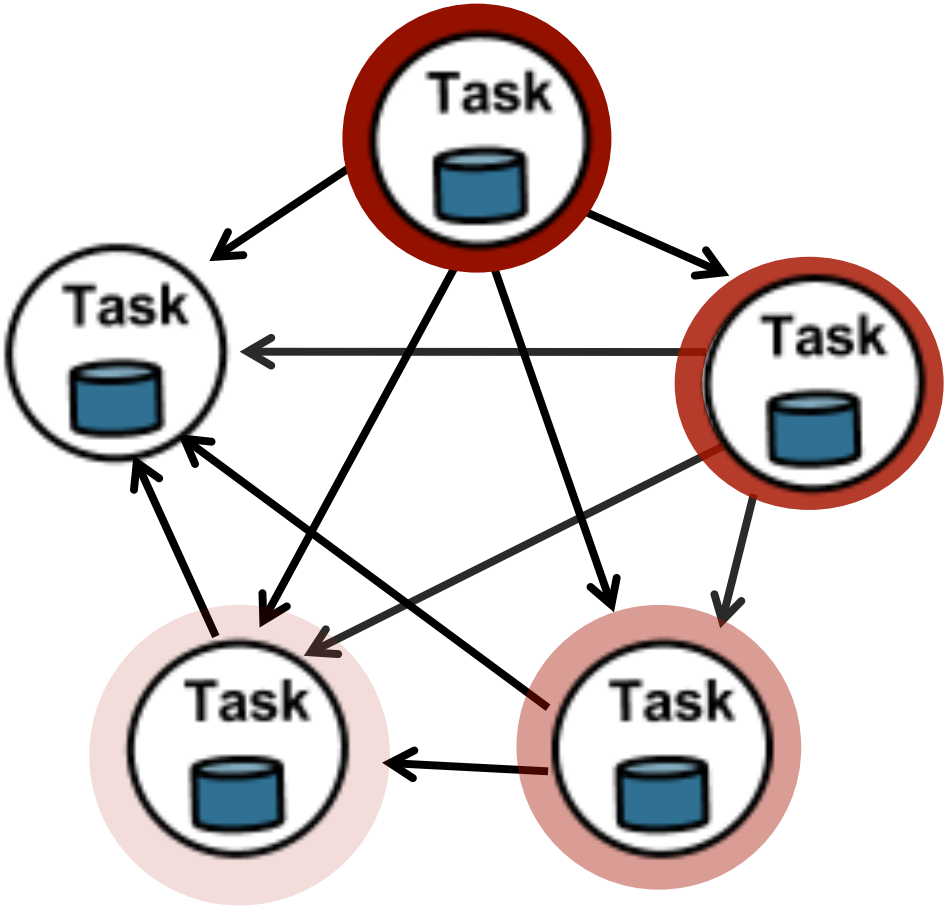


- **Q**: Should P_i compare with P_j or P_j compares with P_i?
  - Impact communication/workload balance.

UCSB

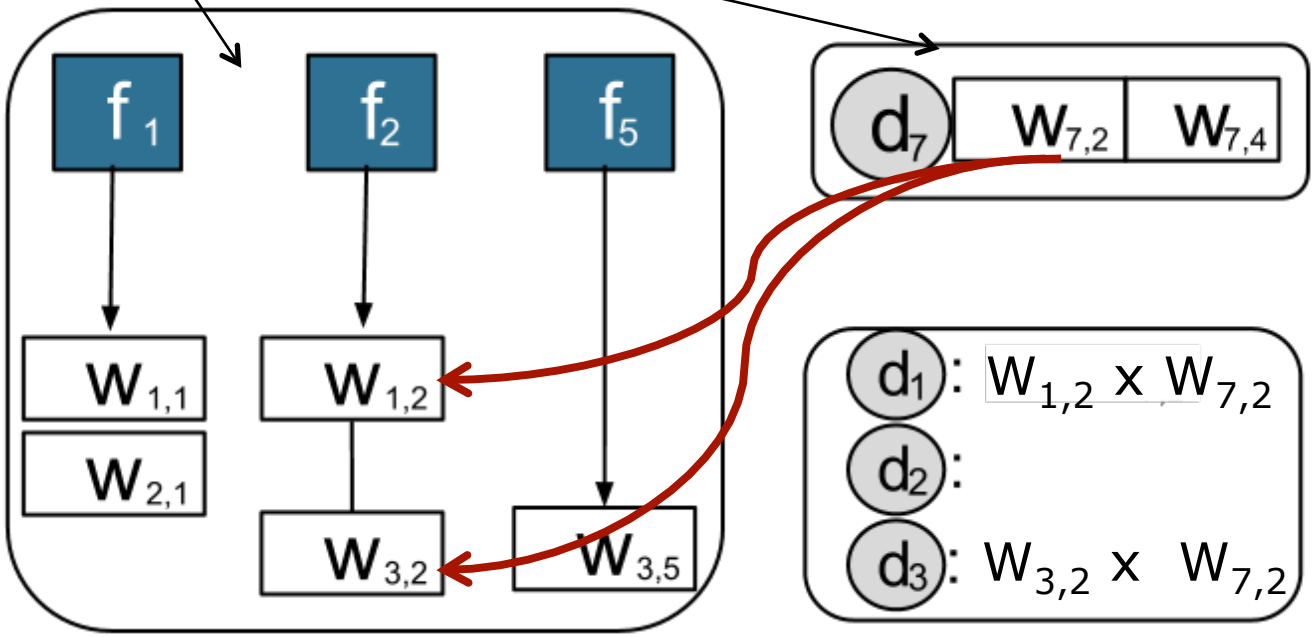
II. Partition-level Circular Load Balancing

- **Goal:**
  - Select partition-level comparison direction.
  - Balance I/O traffic and workload among tasks.



III. Run PSS tasks in parallel

- Simple task parallelism: Do not exploit partial result parallelism.
  - No reduce tasks → no map-reduce communication.
- Hybrid indexing for faster memory data access:
  - Inverted indexing for assigned partition
  - Forward indexing for other vectors



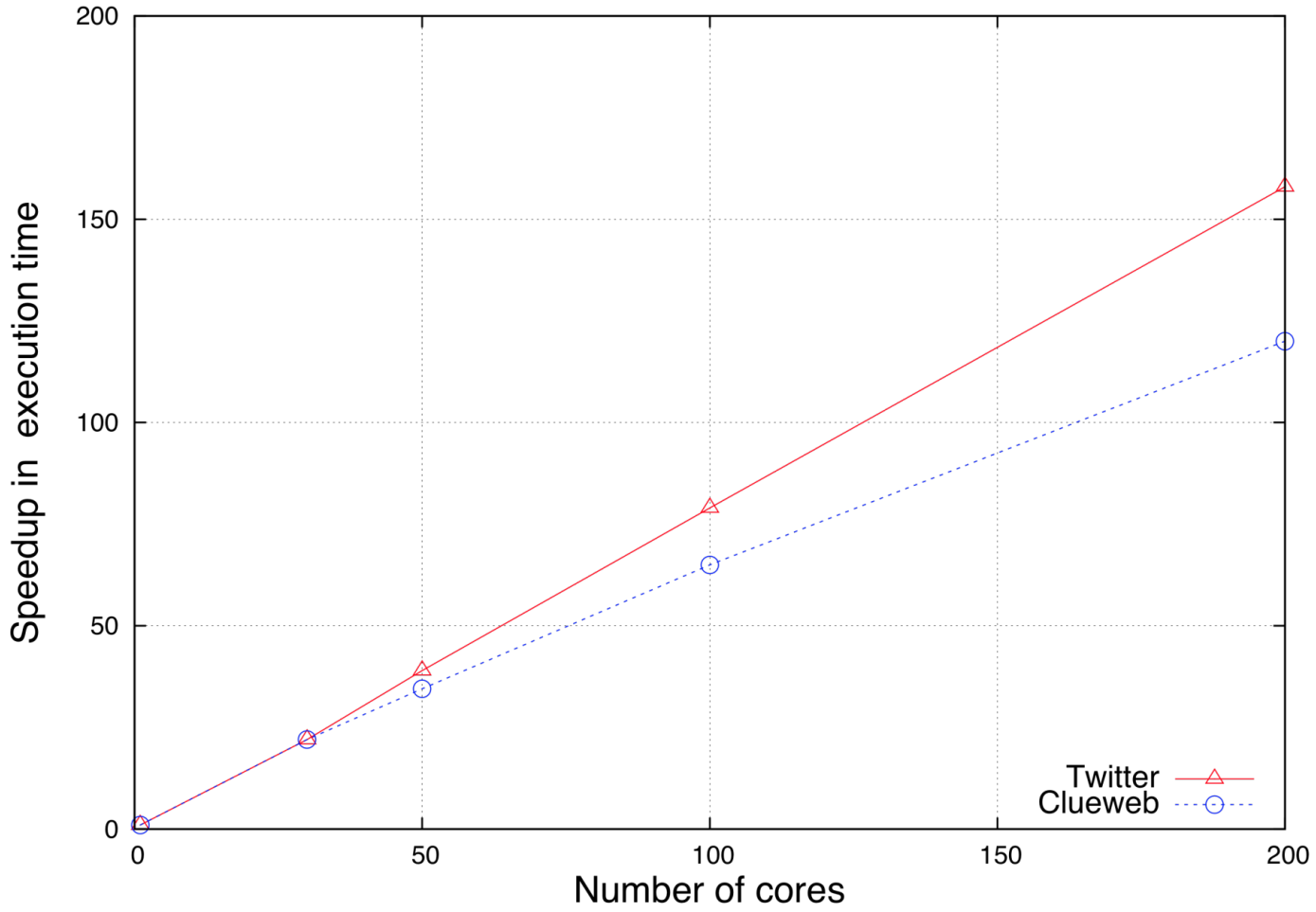
UCSB

Implementation and Evaluation

- **Implemented** in Java with Hadoop MapReduce.
- **Evaluation objectives**:
  - Scalability of PSS with hybrid indexing.
  - Compare against inverted indexing, PSS with forward indexing
  - Effectiveness of static partitioning and partition-based circular balancing.
- **Datasets:** Twitter (20M). Clueweb (50M subset).
  Enron emails (448K).
  - Data cleaning and stopword removal applied first
- Static partitioning takes ~3% of total time.

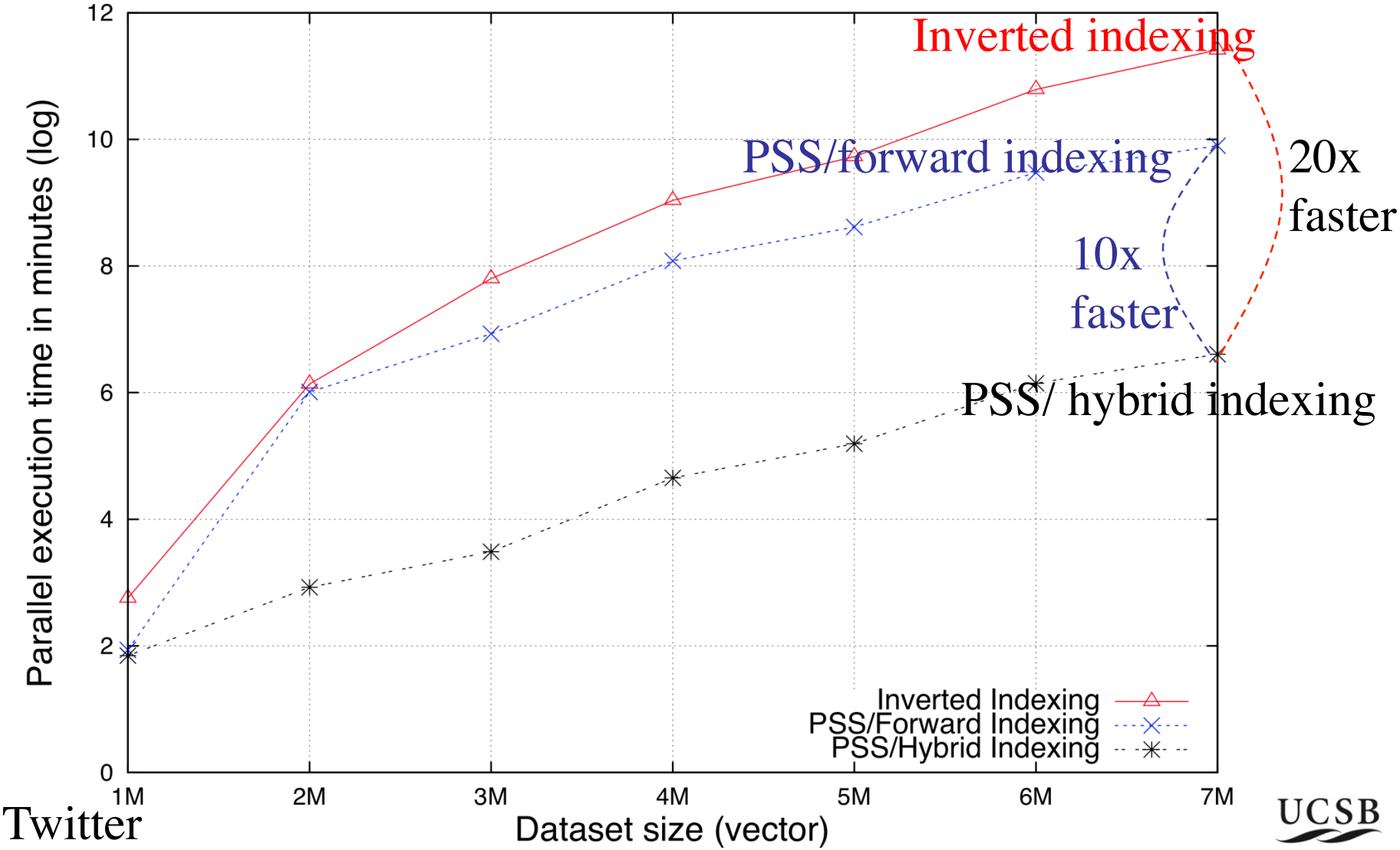
UCSB

Speedup of PSS with hybrid indexing as # of cores increases



UCSB

Parallel time comparison of 3 algorithms



Twitter

UCSB

Ratio of PSS/hybrid indexing time over forward indexing

- p_s : average posting length in the assigned partition
- s: number of vectors in the assigned partitions

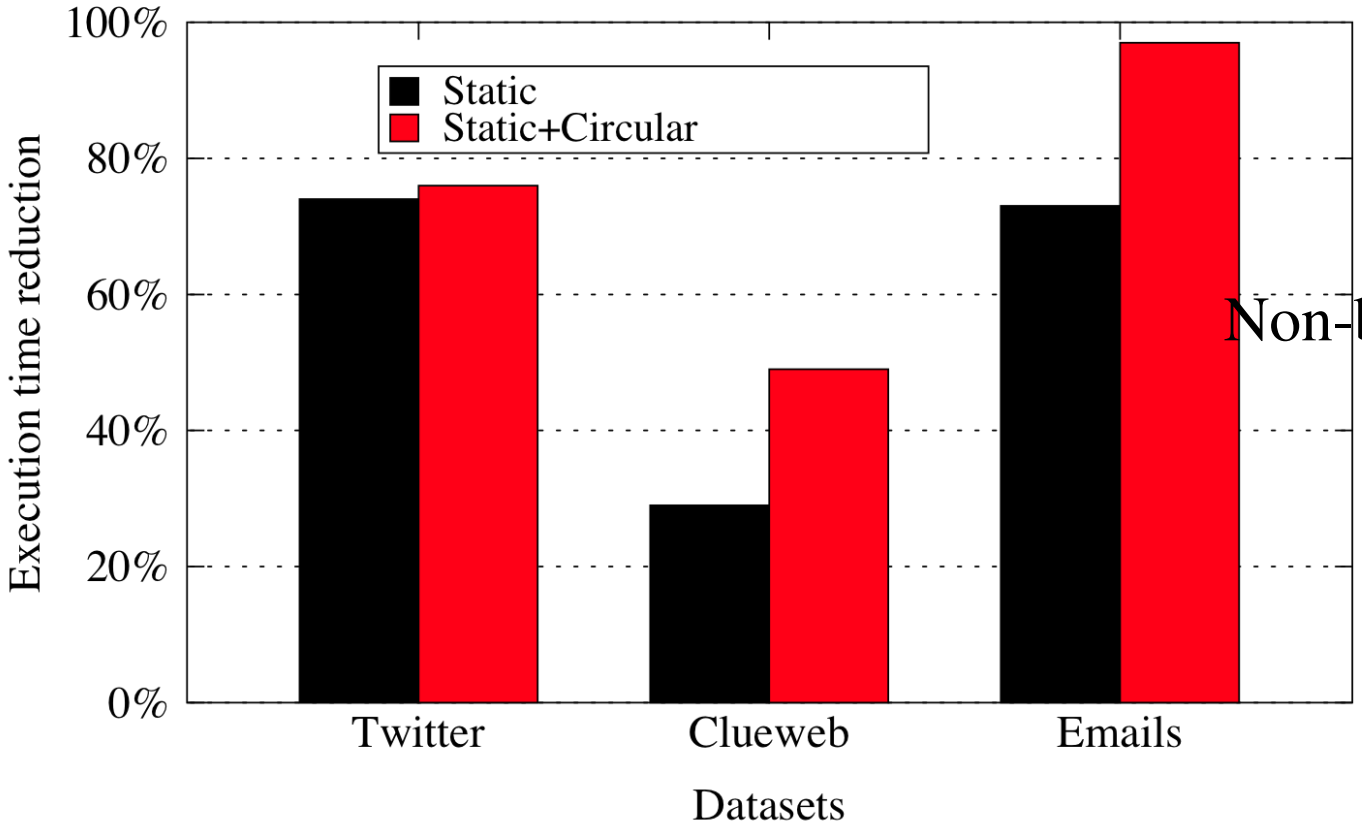
$$\frac{\mathbf{T}_{\mathrm{HI}}}{\mathbf{T}_{\mathrm{FI}}} \approx \frac{l + 4p_s\delta + 4p_s\psi}{2s\delta + 4p_s\psi} \approx \frac{2p_s}{s}$$

- Analysis predicts the trend of cost ratio and impact of faster data access with hybrid indexing.

For Twitter: 20% vs. actual: ~10%

Execution time reduction by static partitioning and partition-level circular balancing

Ratio: 1 – Time(optimized)/Time(baseline)



Non-binary weights

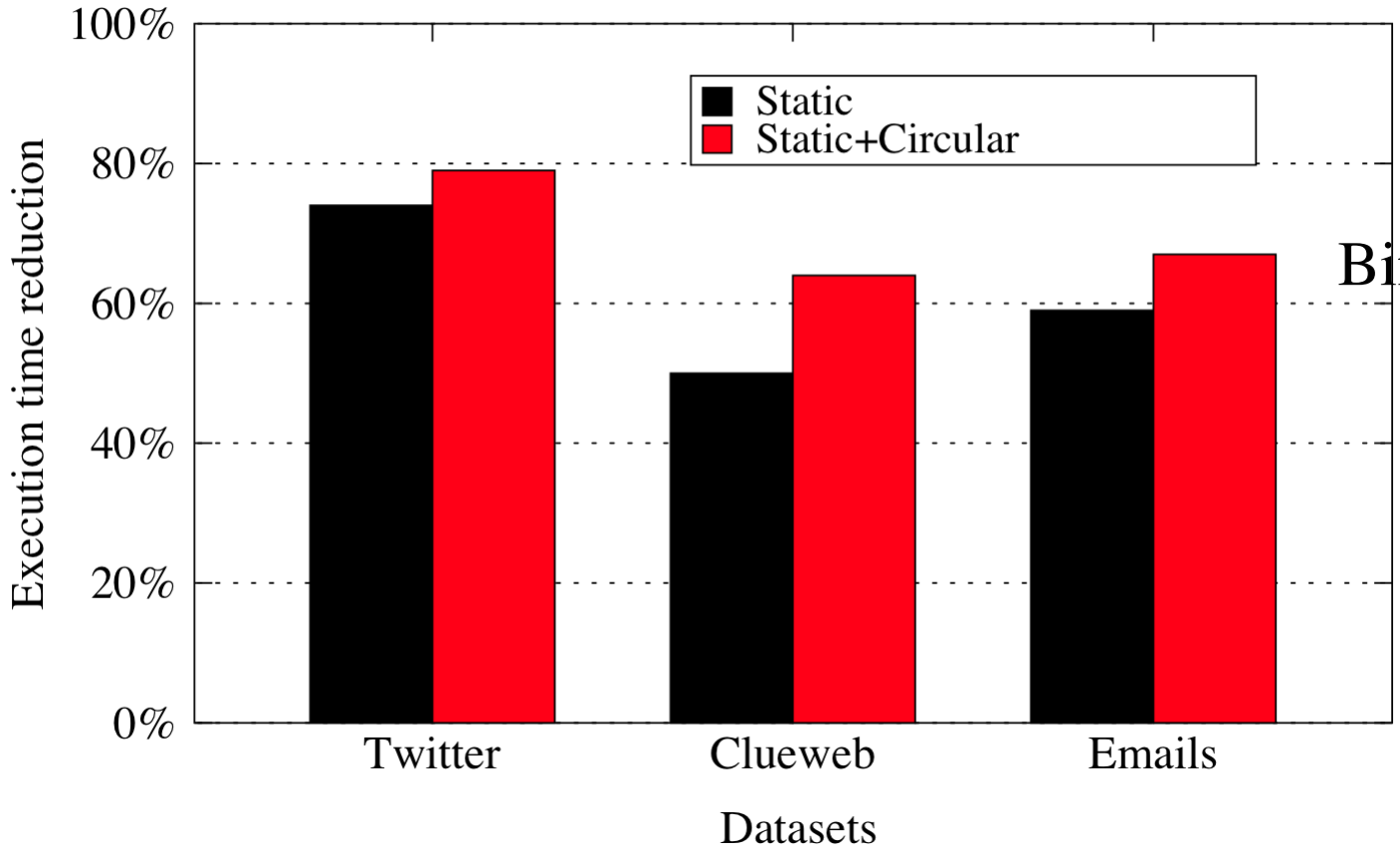
Static partitioning yields 30-75% reduction

Circular balancing adds 3%-26% more

UCSB

Execution time reduction with binary weights

Ratio: 1 – Time(optimize)/Time(baseline)



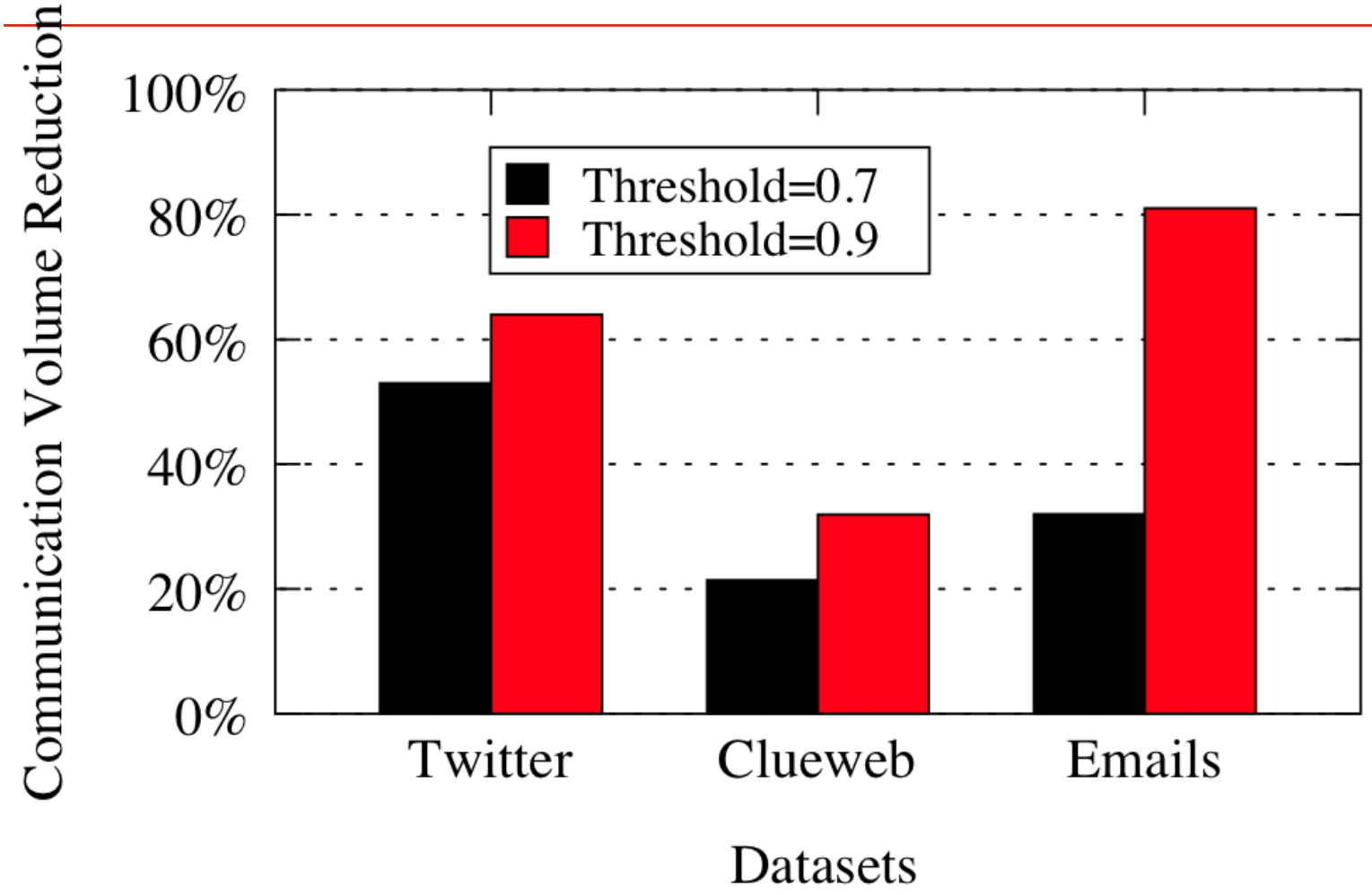
Binary weights

Static partitioning yields 50-75% reduction

Circular balancing adds 4%-11% more

UCSB

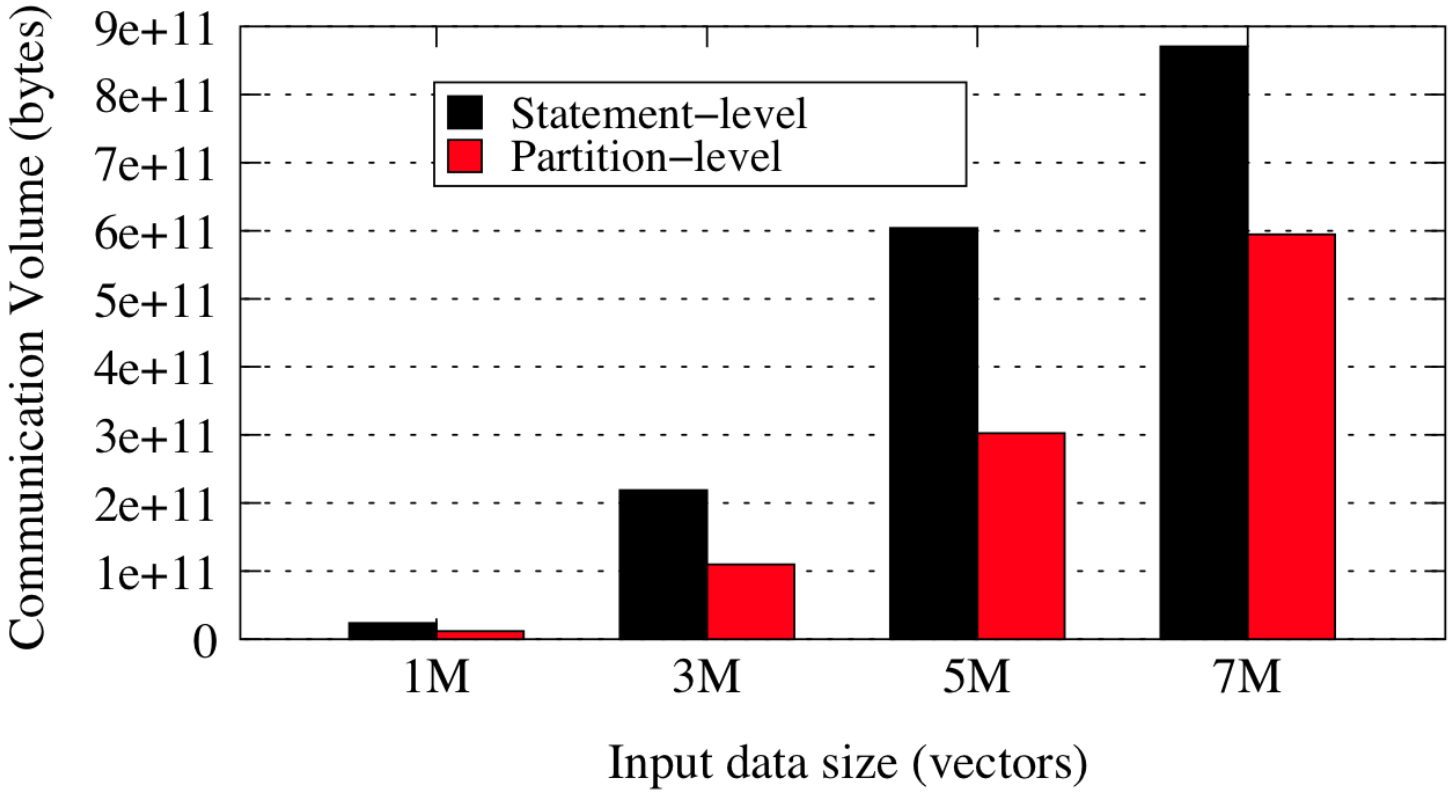
Read I/O Volume Reduction by Static Partitioning



Static partitioning yields 20%-80% volume reduction

UCSB

Read I/O volume of statement-level vs. partition-level symmetric comparison



Twitter

Partition-level symmetric comparison has 50% less read I/O volume

UCSB

Conclusions

- **A scalable two-step approach: partition-based all-pairs similarity search**
  1) fast static partitioning to place dissimilar vectors into different groups.
  2) circular partition-based comparison assignment that exploits computation symmetry & balances workloads.
  3) Simple parallel task execution with hybrid indexing.

- **Up to 10x-20x faster in the tested datasets.**
  - By I/O and communication reduction, & faster memory data access

- **http://www.cs.ucsb.edu/projects/psc**

UCSB