Parallel Data Retrieval in Large Data Sets by Algebraic Methods

Marián Vajteršic

Tobias Berka

University of Salzburg, Austria

Outline

1. Motivation
2. Vector Space Model
3. Dimensionality Reduction
4. Data Distribution
5. Parallel Algorithm
6. Evaluation
7. Discussion

1. Motivation: Automated Information Retrieval

- Problems of *scale*: 500+ million Web pages on Internet, typical search engine updates $\approx$ 10 million Web pages in a single day and the indexed collection of the largest search engine has $\approx$ 100 million documents.
- Development of *automated* IR techniques: processing of large databases without human intervention (since 1992).
- Modelling the concept–association patterns that constitute the *semantic structure* of a document (image) collection (*not* simple word (shape) matching).

1. Motivation: Our Goal

- Retrieval in **large** data sets (*texts, images*)
  - in the **parallel/distributed** computer environment,
  - using **linear algebra** methods,
  - adopting the **vector space** model.
  - in order to get **lower** *response* time and **higher** *throughput*.
- Intersection of three substantially large *IT* fields:
  - *information retrieval* (mathematics of the retrieval models, query expansion, distributed retrieval, etc)
  - *parallel and distributed computing* (data distribution, communication strategies, parallel programming, grid-computing, etc.)
  - *digital text and image processing* (feature extraction, multimedia databases, etc).

2. Vector Space Model: Corpus Matrix

- Documents d_i are vectors in m features

$$d_i = \begin{pmatrix} d_{1,i} \\ \vdots \\ d_{m,i} \end{pmatrix} \in \mathbb{R}^m.$$

- *Corpus matrix* C contains n documents (column-wise)

$$C = \begin{bmatrix} d_1 & \cdots & d_n \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

2. Vector Space Model: Corpus Matrix – Texts versus Images

- Text retrieval: many documents (e.g. 10 000), many terms but FEW terms for each document, hence SPARSE *corpus* matrix.
- Image retrieval: many images, few features (e.g. 500) but FULL feature set for each document, hence DENSE *corpus* matrix.
- DENSE feature vectors of a particular research interest, because
  - dimensionality reduction creates dense vectors
  - multimedia retrieval uses dense vectors
  - retrieval on dense vectors is expensive
  - no proper treatment in literature.
- In both cases (texts, images): the selection of terms (features) is heavily task–dependent.

2. Vector Space Model: Query Matching

- For computing the distance of a query–vector $q \in \mathbb{R}^m$ to the documents, we use cosine similarity:

$$\mathrm{sim}\,(q, d_i) := \cos\,(q, d_i) = \frac{\langle q, d_i \rangle}{\|q\| \|d_i\|} \ .$$

- Using matrix-vector multiplication, we can write

$$\mathrm{sim}\,(q, d_i) = \frac{\left(q^T C\right)_i}{\|q\| \|d_i\|} \ .$$

2. Vector Space Model: Conducting Queries

- In terms of computation:
  - Compute similarity of q for all documents d_i.
  - Sort the list of similarity values.
- In terms of algorithms:
  - First: Matrix-vector product.
  - Then: Sort.
- In terms of costs:
  - Complexity $O(mn)$.
  - 4 GiB $\approx$ 1 million documents with 1024 features (single precision).

2. (Basic) Vector Space Model – Summary

SUMMARY:

- Simple to construct (*corpus* matrix) and conduct queries (cosine similarity).
- Square complexity (one query).
- High memory consumption.
- Sensitivity to failure (e.g. for polysemy and synonyms).

REMEDY:

- Dimensionality reduction (reduction of memory and computational complexity, better retrieval performance).
- Parallelism (speedup of computation, data distribution across memories).
- Most advantageous: a combination of both approaches.

3. Dimensionality Reduction: Goal and Methods

GOAL:

To reduce the dimensionality of the *corpus* without decreasing the retrieval quality

METHODS:

- QR Factorization
- Singular Value Decomposition (SVD)
- Covariance matrix (COV)
- Nonnegative Matrix Factorization (NMF)
- Clustering .

3. Dimensionality Reduction: Formalism

- Assuming we have a matrix L containing k row vectors of the length m.
- We project every column of C on all k vectors, using the matrix product LC.
- Projection-based dimensionality reduction can be seen as as a linear function

$$f(v) = Lv \ (v \in \mathbb{R}^m)$$

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^k, \ k < m.$$

3. Dimensionality Reduction: QR

- Compute the decomposition $C = QR$, where Q of the size $m \times m$ is orthogonal ($QQ^T = Q^TQ = I$) and R (size $m \times n$) is upper triangular.
- If rank$(C) = r_C$, then r_C columns of Q form a basis for the column space of C.
- QR factorization with complete column pivoting (i.e., $C \rightarrow CP$ where P is the permutation matrix) gives the column space of C but **not** the row space.
- QR factorization enables to decrease the rank of C but **not** optimally.

3. Dimensionality Reduction: SVD

- $C = U\Sigma V^T$ ... singular value decomposition of C
- $C \approx U_k \Sigma_k V_k^T$ ... rank-k approximation
- $C' = U_k^T C$ ... reduced corpus
- $q' = U_k^T q$ ... reduced query.
- SVD of C: both column **and** row spaces of C are computed and ensures the optimal value of k for decreasing the rank.

3. Dimensionality Reduction: SVD

OUR COMPETENCE:

- Parallel block-Jacobi SVD algorithms.
  Our approach with the dynamic ordering and preprocessing performs for some matrix types better than SCALAPACK (Bečka, Okša, Vajteršic; 2010).
- Application of (parallel) SVD to Latent Semantic Indexing (LSI) Model (Watzl, Kutil; 2008).
- Parallel SVD Computing in the Latent Semantic Indexing Applications for Data Retrieval (Okša, Vajteršic; 2009).

3. Dimensionality Reduction: COV

- Compute the covariance matrix of C.
- Compute the eigenvectors of the covariance matrix.
- Assume E_k are the k largest eigenvectors (column-wise), then
  - $C' = E_k^T C$ ... reduced corpus,
  - $q' = E_k^T q$ ... reduced query.

3. Dimensionality Reduction: NMF

MOTIVATION:

- Corpus matrix C is nonnegative.
- However, SVD cannot maintain nonnegativity in the low–rank approximation (because the components of left and right singular vectors can be negative).
- When aiming to preserve the nonnegativity also in the k-rank approximation, we have to apply NMF.

NMF:

- For a positive integer $k < \min(m, n)$ compute nonnegative matrices $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$.
- The product WH is a nonnegative matrix factorization of C (although C is not necessarily equal to WH) but it can be interpreted as a compressed form of C.

3. Dimensionality Reduction: NMF

BASIC COMPUTATIONAL METHODS for NMF:

- ADI Newton iteration
- Multiplicative Update Algorithm
- Gradient Descent Algorithm
- Alternating Least Squares Algorithms.

OUR COMPETENCE:

- Nonnegative Matrix Factorization: Algorithms and Parallelization. (Okša, Bečka, Vajteršic; 2010)
- FWF project proposal (Parallelization of NMF) with Prof. W. Gansterer, University of Vienna (in preparation).

3. Dimensionality Reduction: Clustering

- Compute k clusters of the column vectors of C.
- Compute a representative vector for every cluster.
- Assume R are the representatives (column-wise), then
  - $C' = R_k^T C$ ... reduced corpus
  - $q' = R_k^T q$ ... reduced query.

OUR COMPETENCE:

- Analysis of clustering approaches (Horak; 2010)
- Parallel Clustering Methods for Data Retrieval. (Horak, Berka, Vajteršic; 2010 (in preparation))

4. Data Distribution: Partitionings

GOAL:
To reduce the dimensionality of the *corpus* matrix through its partitioning into submatrices for parallel execution.

- Feature partitioning – vertical partitioning: row partitioning.
- Document partitioning – horizontal partitioning: column partitioning.
- Hybrid partitioning – combines both: block partitioning.

4. Data Distribution: Row Partitioning

- Split the features F into M sub-collections,

$$F = \coprod_{i=1}^{M} F_i,$$

- and split the corpus matrix horizontally

$$C = \begin{bmatrix} C[1] \\ \vdots \\ C[M] \end{bmatrix},$$

- into local corpus matrices

$$C[i] \in \mathbb{R}^{m_i \times n}.$$

4. Data Distribution: Column Partitioning

- Split the documents D into N sub-collections,

$$D = \coprod_{i=1}^{N} D_i,$$

- and split the corpus matrix vertically

$$C = [C[1] \cdots C[N]],$$

- into local corpus matrices

$$C[i] \in \mathbb{R}^{m \times n_j}.$$

4. Data Distribution: Block Partitioning

- Split the corpus matrix block-wise

$$C = \begin{bmatrix} C[1,1] & \cdots & C[1,N] \\ \vdots & \ddots & \vdots \\ C[M,1] & \cdots & C[M,N] \end{bmatrix},$$

- into NM local corpus matrices

$$C[i,j] \in \mathbb{R}^{m_i \times n_j}.$$

4. Data Distribution: Example Block Distribution




IMAGE CORPUS:
1024 amateur color photographs from different landscapes:
arctic, alpine, beach shores, desert.
320×320 pixels, into 32×32 blocks with 512 features each (3D histogram)

5. Parallel Algorithm: Potential of Parallelism

- Algebraic Methods for the IR problem are good candidates for efficient parallelization.
- Exploitation of many processors enables to reduce the computational and memory complexity.
- Parallelism can be applied on more hierarchical levels of the solution of the problem.

OUR COMPETENCE:

- 40 years experience in development of parallel algorithms and programs.
- EU, NATO and CEI projects in the area of parallel computing.
- AGRID national prject in Grid-computing.
- Trobec, R., Vajteršic, M., Zinterhof, P. (Eds.): Parallel Computing: Numerics, Applications, and Trends. SPRINGER-Verlag, London, 2009.

5. Parallel Algorithm: Characteristics

- Dimensionality–reduction and query–processing on dense vectors in the basic vector space model.
- Target architecture: parallel computer with distributed memory.
- Infrastructure: cluster system.
- Programming paradigm: message passing with MPI.
- Programming language: C++, C, FORTRAN.

5. Parallel Algorithm: Node Organization

- 2D mesh of the size $P = M \times N$.
- Set of features per row.
- Set of documents per column.
- Every node holds a block $C(i, j)$ $(i = 1, ..., M;\ j = 1, ..., N)$ of the corpus matrix (block–partitioning).
- Goal: exploit nested parallelism with rows and columns.

5. Parallel Algorithm: Dimensionality–Reduction

Dimensionality reduction LC using $L \in \mathbb{R}^{k \times m}$:

- Split L into

$$L = \begin{bmatrix} L[1] \\ \vdots \\ L[M] \end{bmatrix},$$

- with local projection matrices

$$L[i] \in \mathbb{R}^{k_i \times m}.$$

- Distribute $L[i]$ to all processing nodes in the i-th row.
- Distribute the j-th block-column of C to all nodes in the j-th row.
- On each node (i, j) compute the reduction $L[i]C[*, j]$ locally.
- Theoretic speed-up $O(NM)$.

5. Parallel Algorithm: Query Matching

- Distribute the (row)query–vector q to all nodes.
- On each node (i, j) compute the matrix-vector product locally:

$$(qC)_i = \sum_{j=1}^{m} q_j C_{j,i} = \sum_{h=1}^{M} \sum_{j=1}^{m_h} q[h]_j C[h]_{j,i} \ .$$

- All processors in the j-th column of the mesh $(j = 1, ..., N)$ cooperative compute

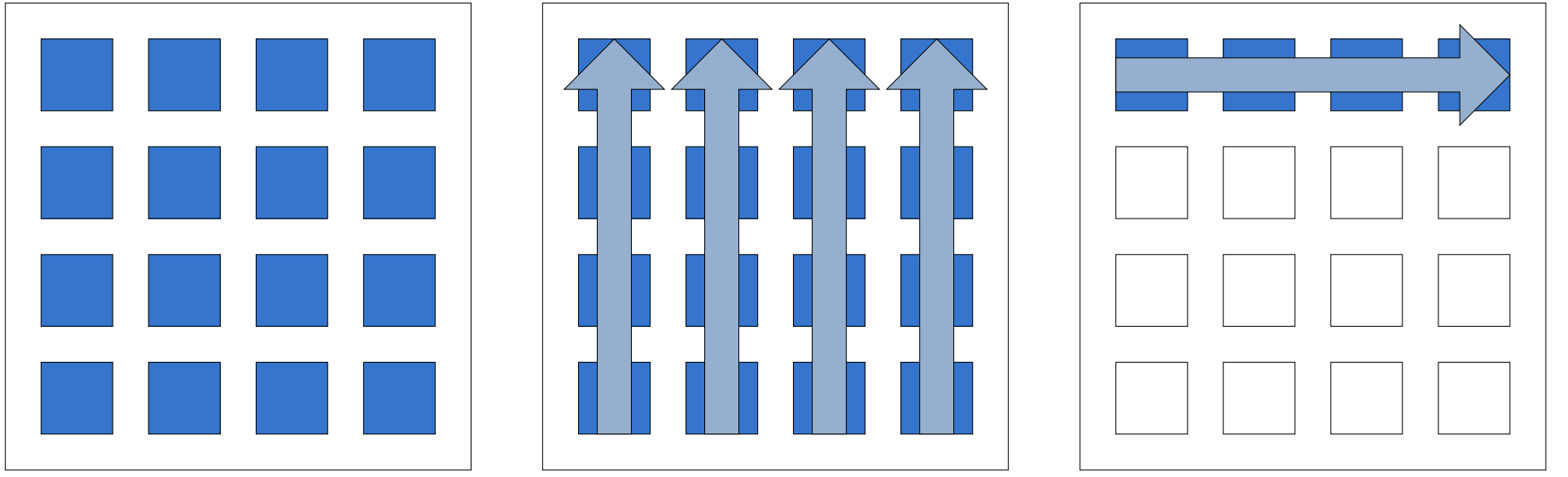
$$r[j] = qC[*, j] = \sum_{i=1}^{M} q[i]C[i, j] \, .$$

- Generate

$$r = qC = ([r[1], r[2], ..., r[N]) \, .$$

- Sort components of r .

5. Parallel Algorithm: Overview

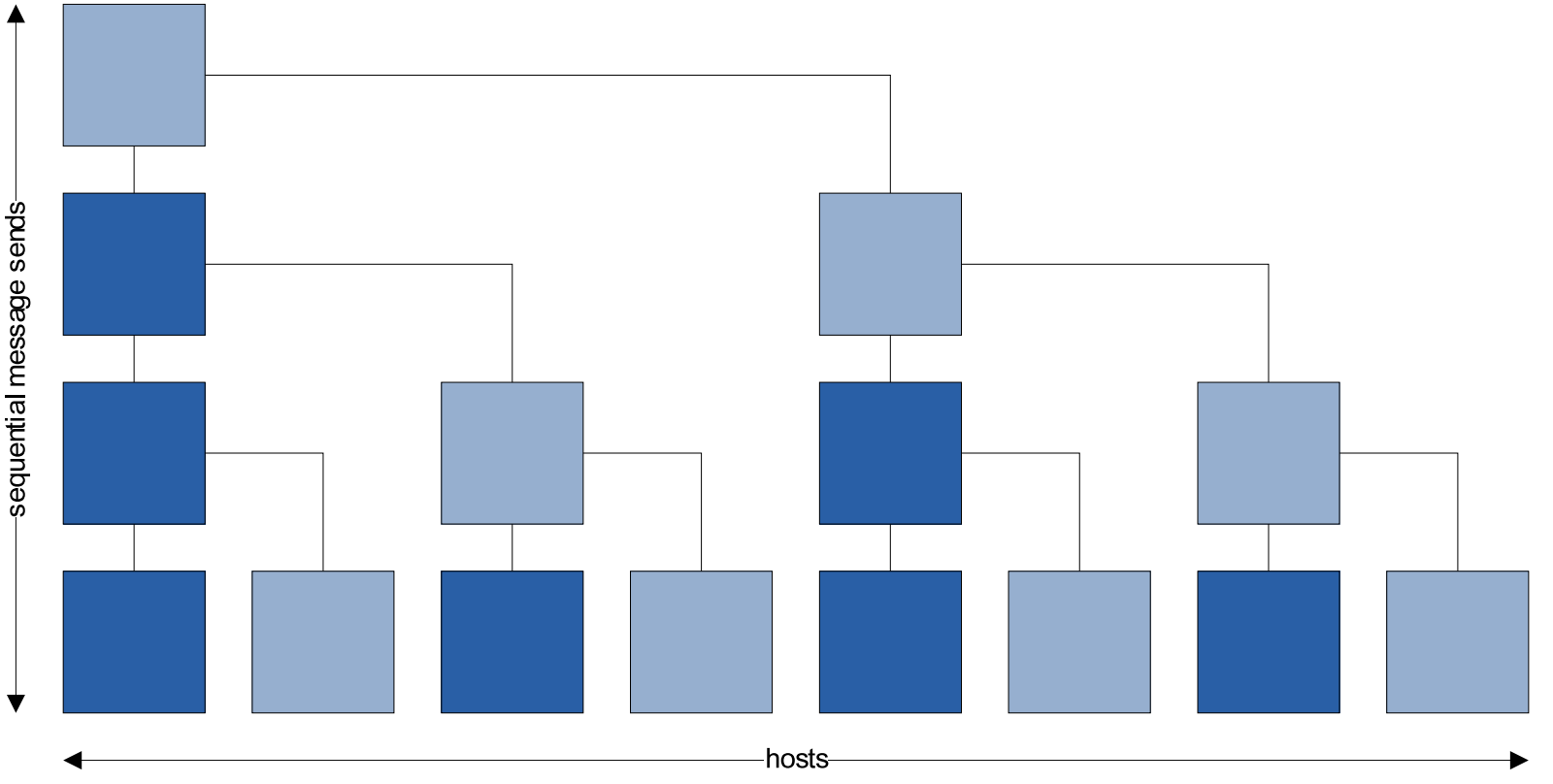


- Broadcast the appropriate query data to all nodes.
- Compute local results.
- Accumulate matrix-vector product.
- Merge-sort result entity-similarity pairs.

5. Parallel Algorithm: MPI

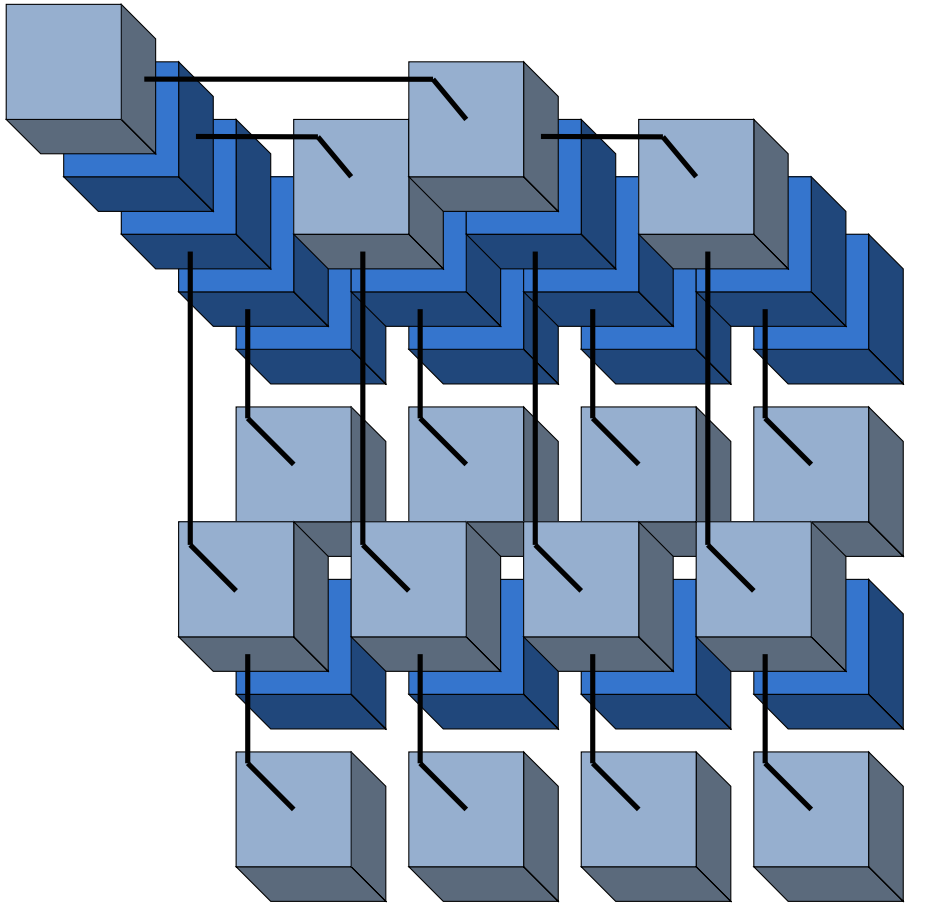
- Distribute query: MPI broadcast.
- Matrix-vector product: using MPI–reduce with the Sum–operator.
- Merge-sort: MPI only provides collective *vector* operations (fixed length).

5. Parallel Algorithm: Merge-Sort Communication Structure



Flat binary tree / hypercube (with the natural binary addressing).

5. Parallel Algorithm: Nested Communication Structure



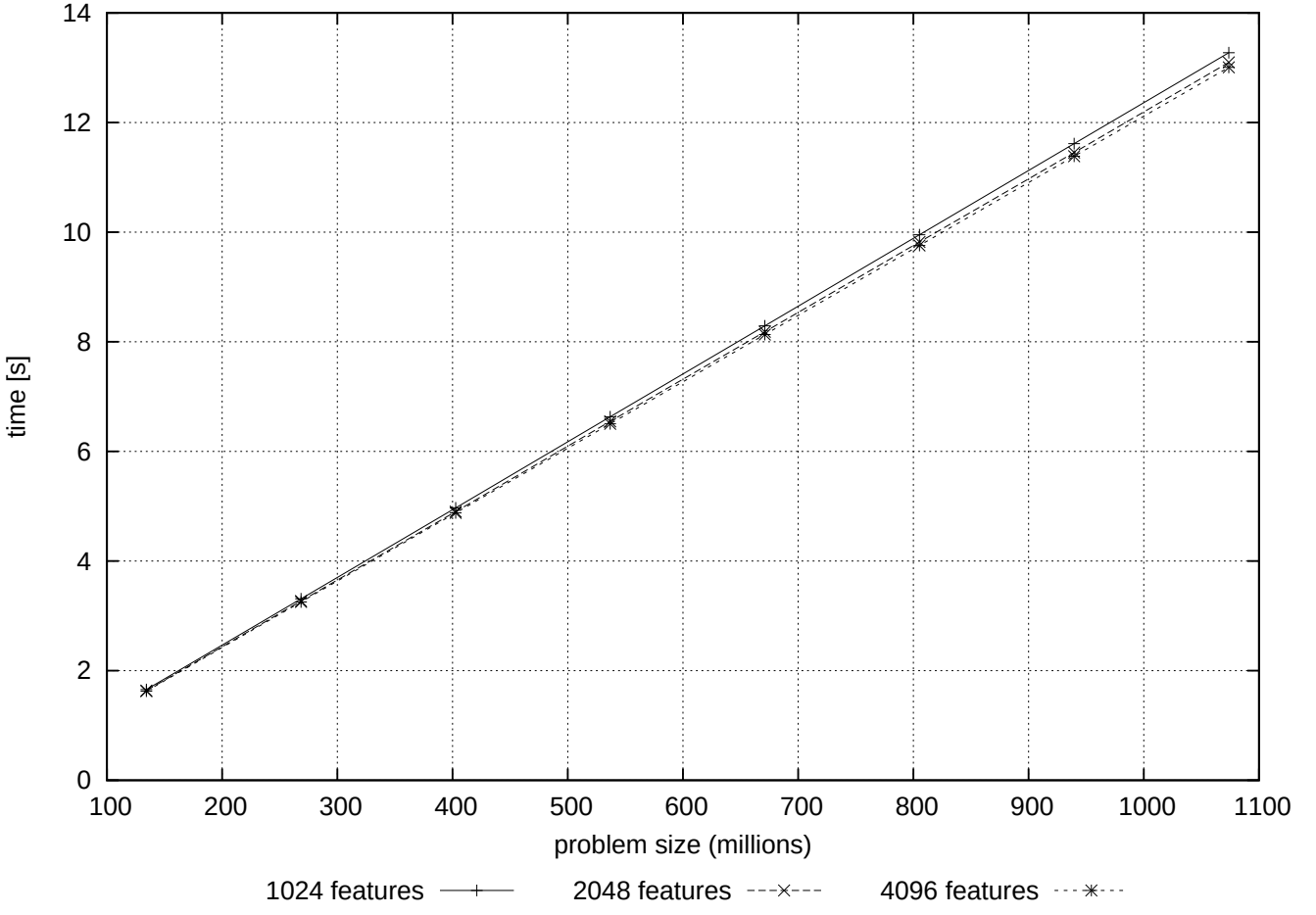
6. Evaluation: Theoretic Speed-up

- Vector matrix product dominates complexity.
- Best case: linear speed-up.
- Balanced distribution of entities (i.e. $m_i = \frac{m}{M}$, $k_i = \frac{k}{M}$ for $i = 1, ..., M$ and $n_j = \frac{n}{N}$ for $j = 1, ..., N$) is important!

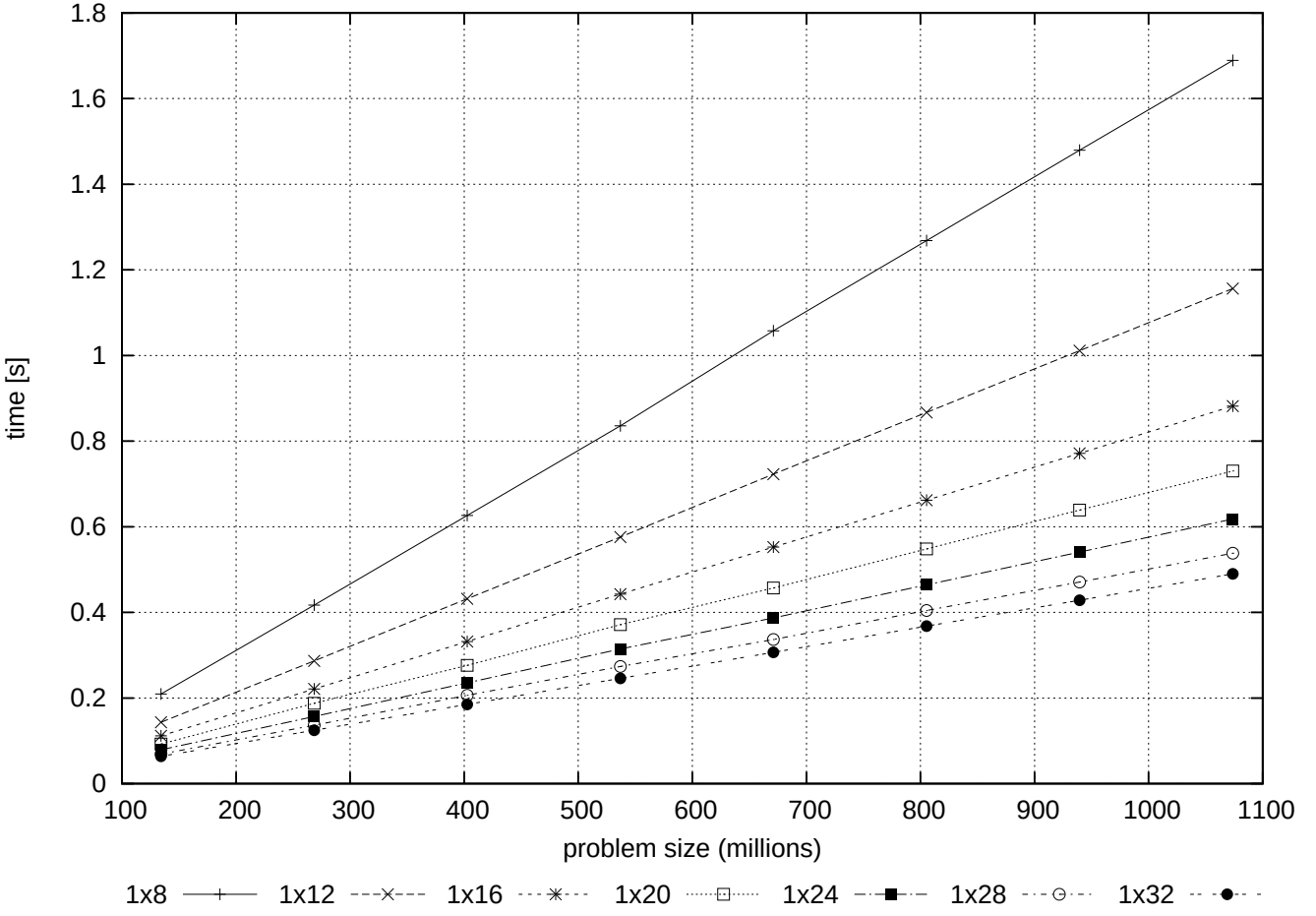
6. Evaluation: Measured Speed-Up

- For 1024 – 4096 features.
- For 100,000 – 1,000,000 documents.
- For all 3 partitioning strategies:
  - pure feature partitioning failed
  - document partitioning provided good efficiency
  - hybrid partitioning delivered super-linear speed-up.
- Recommended topology: $2 \times N$.

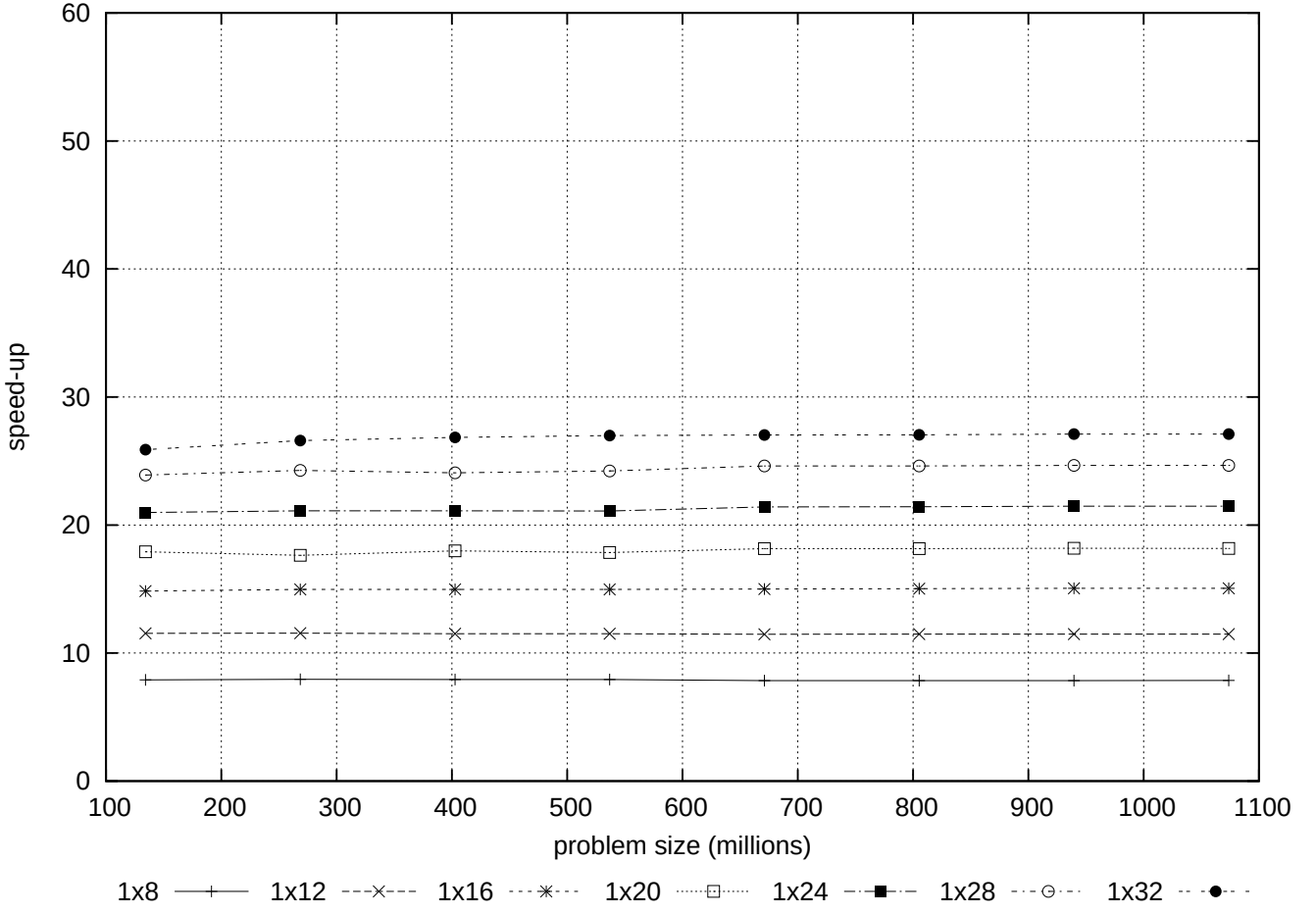
6. Evaluation: Serial Response Time



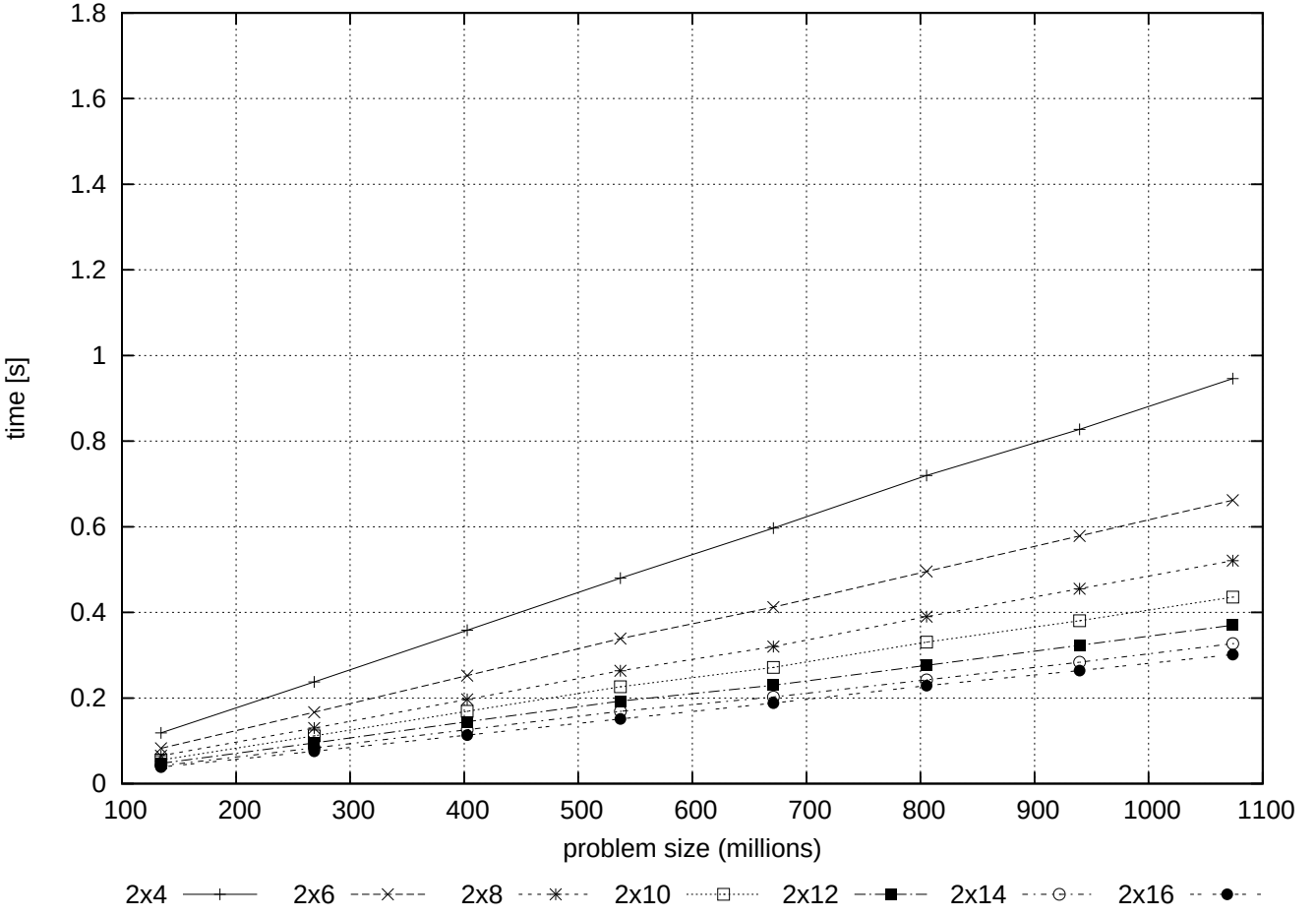
6. Evaluation: Document Partitioning Response Time



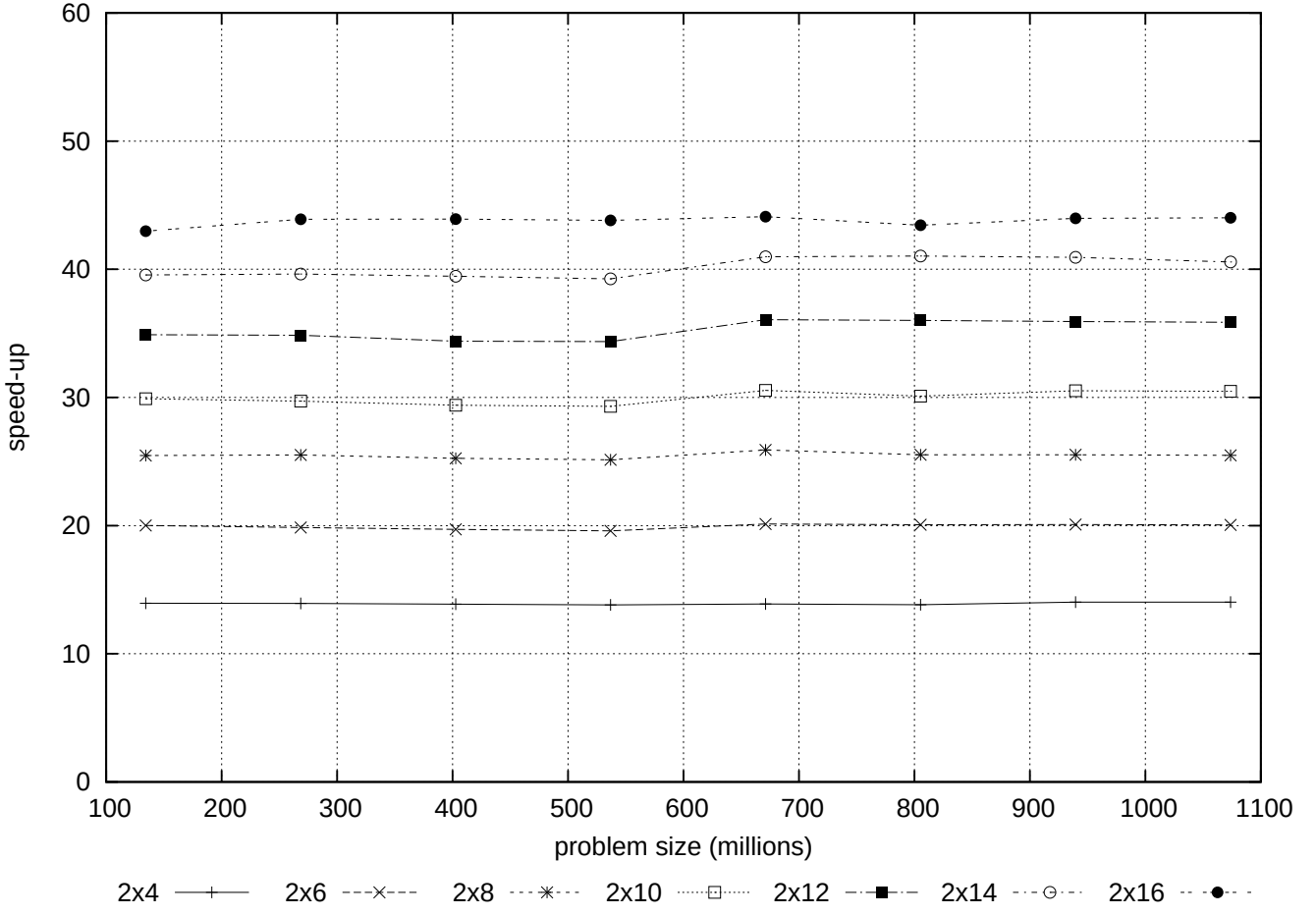
6. Evaluation: Document Partitioning Speed-up



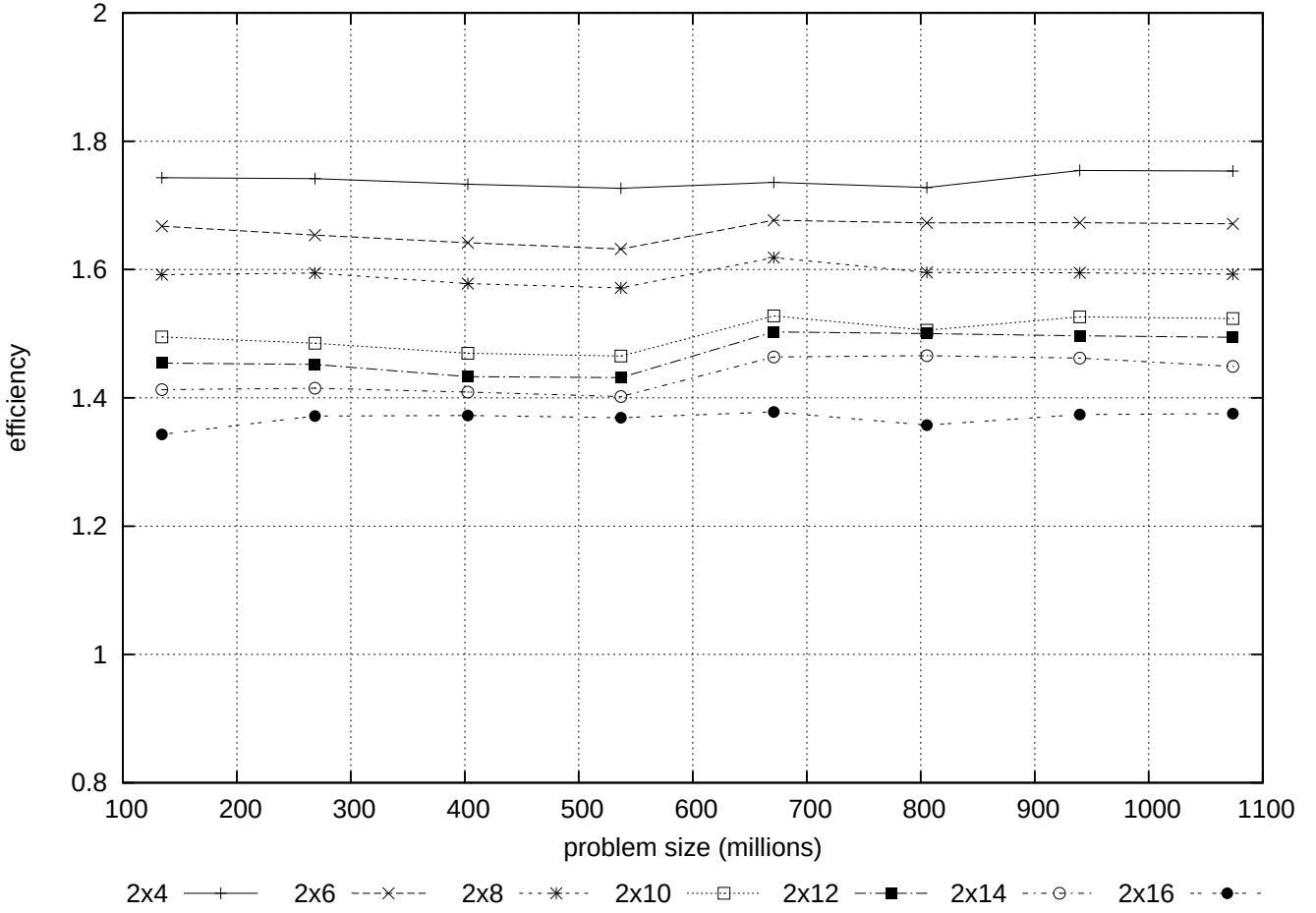
6. Evaluation: Hybrid Partitioning Response Time



6. Evaluation: Hybrid Partitioning Speed-up



6. Evaluation: Hybrid Partitioning Efficiency



6. Evaluation: Serial and Parallel Response Times and Throughputs

- t_s : response time for complete processing one query vector on one processing node (serial response time).
- $T_s = \frac{1}{t_s}$: serial throughput (number of processed queries in 1 sec. on one processor).
- $T_{old} = \frac{NM}{t_s}$: throughput for a naive NM-times replication of the serial computation on NM processors.
- t_p : parallel response time for complete processing of one query vector on NM processors (parallel response time).
- $T_{new} = \frac{1}{t_p}$: throughput for a parallel implementation of the query processing on NM processors.

6. Evaluation: Improvements

- Speed-up $S = \frac{t_s}{t_p} > 1$: improved response time.
- Efficiency $E > 1$: improved throughput.
- Gain in throughput $\frac{T_{old}}{T_{new}}$ is equal to the parallel efficiency E:

$$\frac{T_{new}}{T_{old}} = \frac{\frac{1}{t_p}}{\frac{NM}{t_s}} = \frac{t_s}{t_p}\frac{1}{NM} = \frac{S}{NM} = E\,.$$

7. Discussion

- IR is a concurrent task:
  - add/remove documents
  - update and downdate operations
  - multi-user operation.
- IR is a long-term activity:
  - checkpointing?
  - partial recovery?
- Study further mechanisms:
  - caching
  - clustering
  - parallel programming paradigms (multithreading,...)
- Construct a complete, parallel high-performance IR system.