Enhancing Similarity Search Performance by Dynamic Query Reordering

Filip Nálepa, Michal Batko, Pavel Zezula

Stream Processing in Similarity Search

- Motivation
  - Image annotation – annotate a stream of images collected by a web crawler
  - Publish/subscribe applications – categorize a stream of documents
  - → stream of query objects
- Stream: potentially infinite sequence of query objects (q_1, q_2, …)
- Process as many query objects as possible, processing of a query object can be delayed → **maximize throughput**

Problem Definition

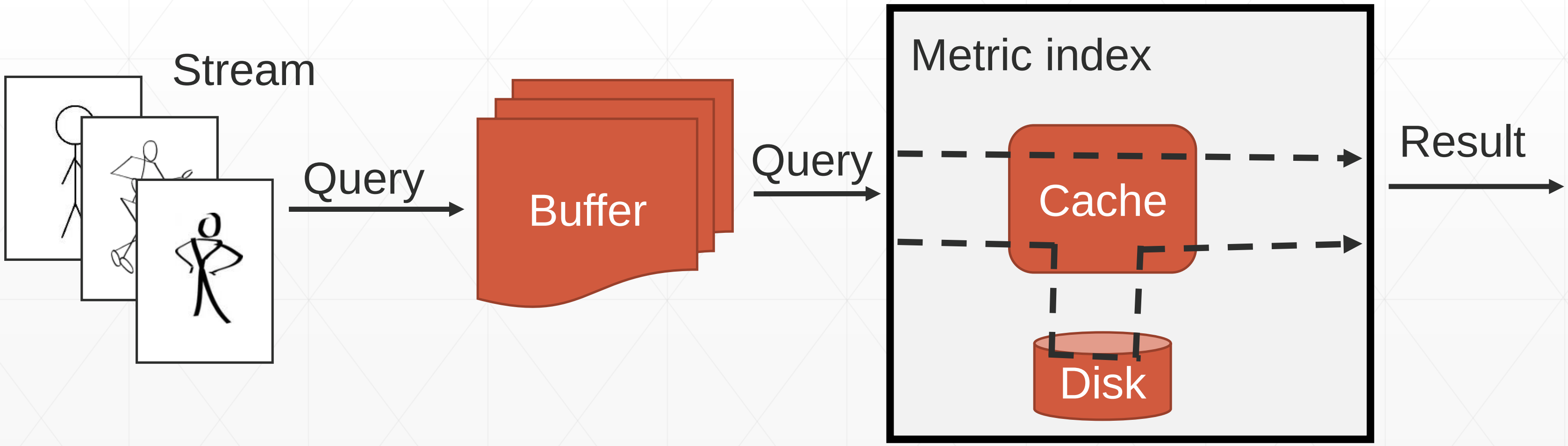
- Domain of objects D
- DB of objects D indexed in the metric space
  - Distance function d: D x D → **R** determines the similarity of two objects
- Stream of query objects $((q_1, t_1), (q_2, t_2), \ldots)$
  - $q_i \in D$
  - t_i – time of arrival, $t_i \leq t_{i+1}$
- Evaluate k-NN query for each q_i, i.e., find k most similar objects in DB to q_i
- Optimization criteria – throughput
  - Maximize the number of processed query objects

Similarity Search Approach

- Typical similarity search techniques:
  - Partitioned data of DB stored on a disk
  - Read a subset of partitions during query evaluation → bottleneck
- Assumption: similar query objects need similar sets of partitions
- Idea: reuse loaded partitions to save disk accesses → data partition caching
- Problem: huge metric space → low probability of data partition intersection
- Solution: reorder query objects to obtain sequences of similar query objects

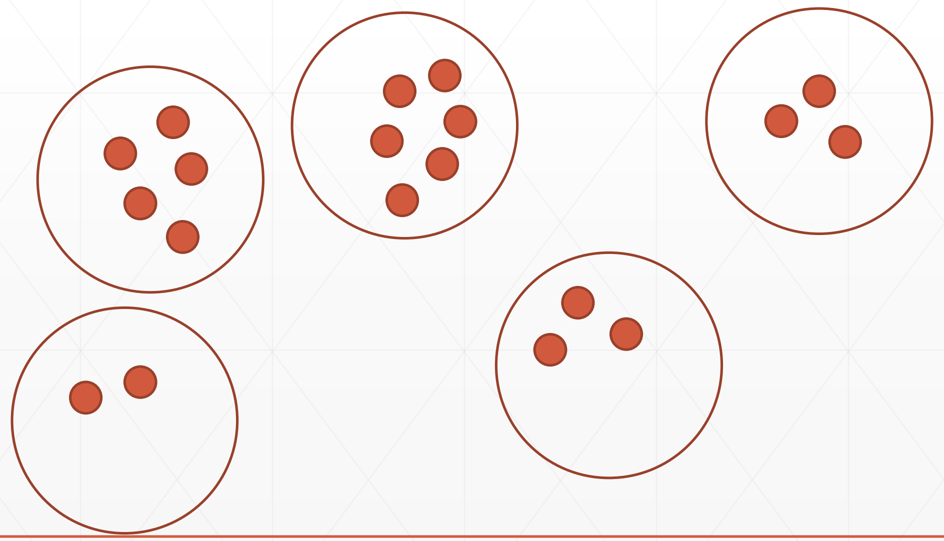
Architecture

- Buffer: waiting query objects, query object reordering
- Metric index: query evaluation
- Cache: in-memory caching of data partitions



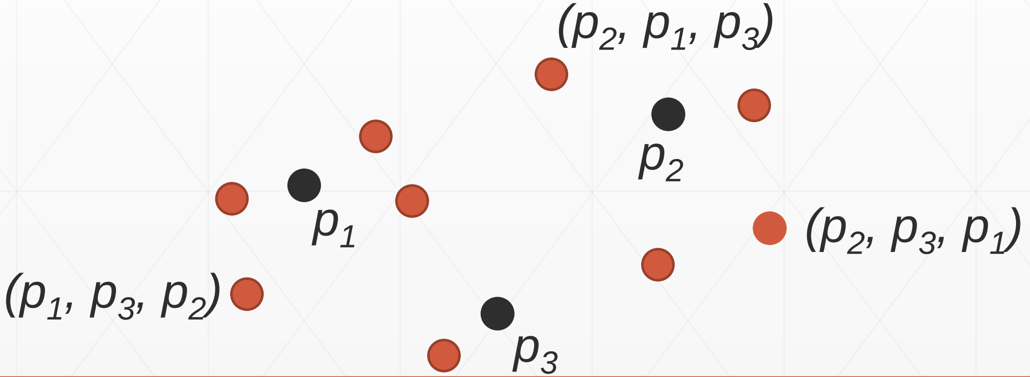
Query Object Reordering within the Buffer

- Task: find sequences of similar query objects
- Solution:
  - cluster query objects
  - select a cluster and evaluate all the query objects in that cluster



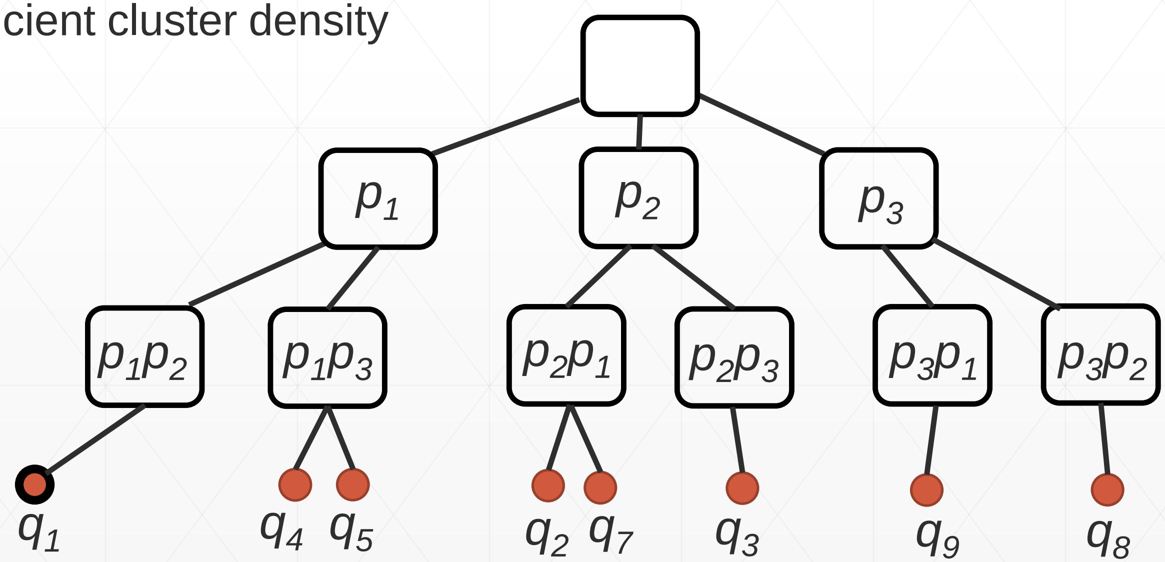
How to Cluster?

- Has to be efficient
- Pivot-based clustering
- Fixed set of pivots $p_1, \ldots, p_n$ in the metric space
- Compute metric distance of a new query object to all the pivots
- Order the pivots from the nearest to the farthest one → pivot permutation = cluster



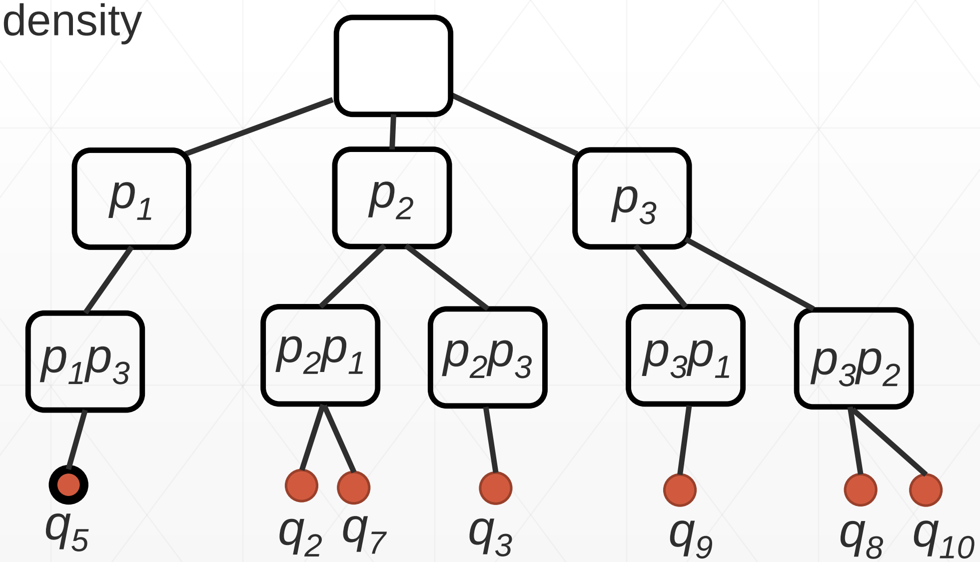
Hierarchical Clustering

- Individual levels correspond to the length of the common pivot permutation prefix
- Internal node – common prefix of all children
- Leaves – query objects
- Query ordering: depth-first tree traversal
  - Find lowest nonempty parent of previous query object → similar cluster
  - Select child containing the oldest query object → no query starvation, sufficient cluster density



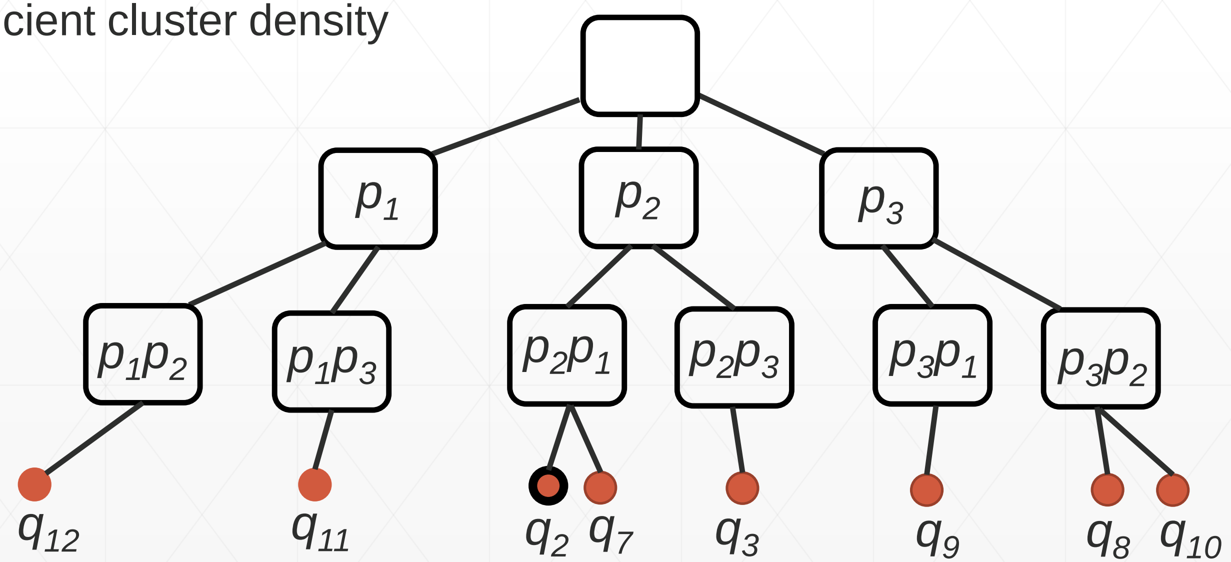
Hierarchical Clustering

- Individual levels correspond to the length of the common pivot permutation prefix
- Internal node – common prefix of all children
- Leaves – query objects
- Query ordering: depth-first tree traversal
  - Find lowest nonempty parent of previous query object → similar cluster
  - Select child containing the oldest query object → no query starvation, sufficient cluster density



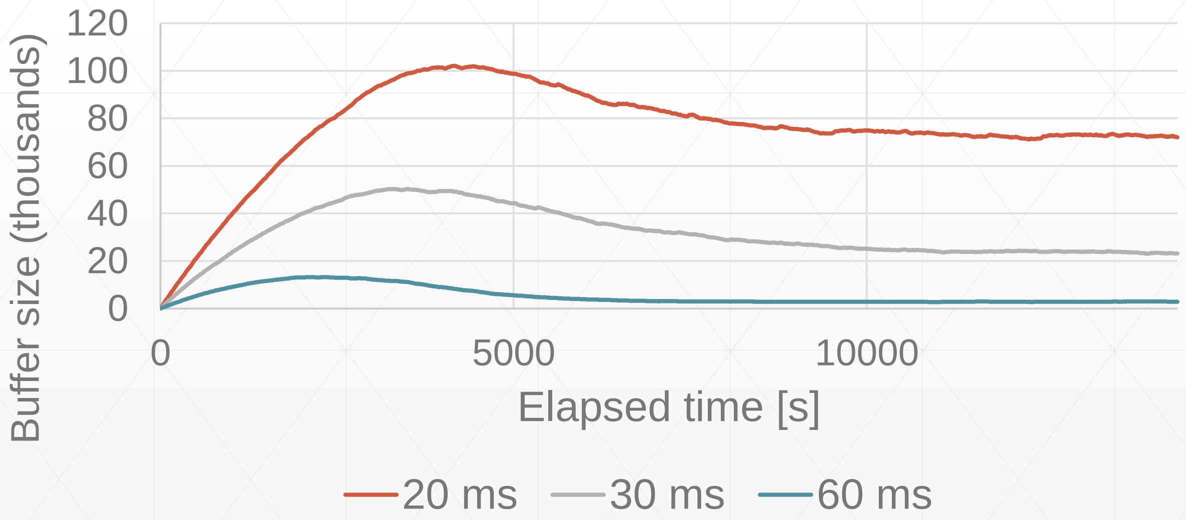
Hierarchical Clustering

- Individual levels correspond to the length of the common pivot permutation prefix
- Internal node – common prefix of all children
- Leaves – query objects
- Query ordering: depth-first tree traversal
  - Find lowest nonempty parent of previous query object → similar cluster
  - Select child containing the oldest query object → no query starvation, sufficient cluster density



Experiments – Fixed Input Rate

- DB: 10 mil. images represented by MPEG-7 descriptors
- Stream of query objects: evaluation of approximate 10-NN queries (10 nearest neighbors)
- Cache size: 90,000 objects (0.9% of the DB)
- Fixed input rate: new query object arrives every x time units
- Average query time for no reordering and no caching: 113 ms



Throughput Delay Tradeoff

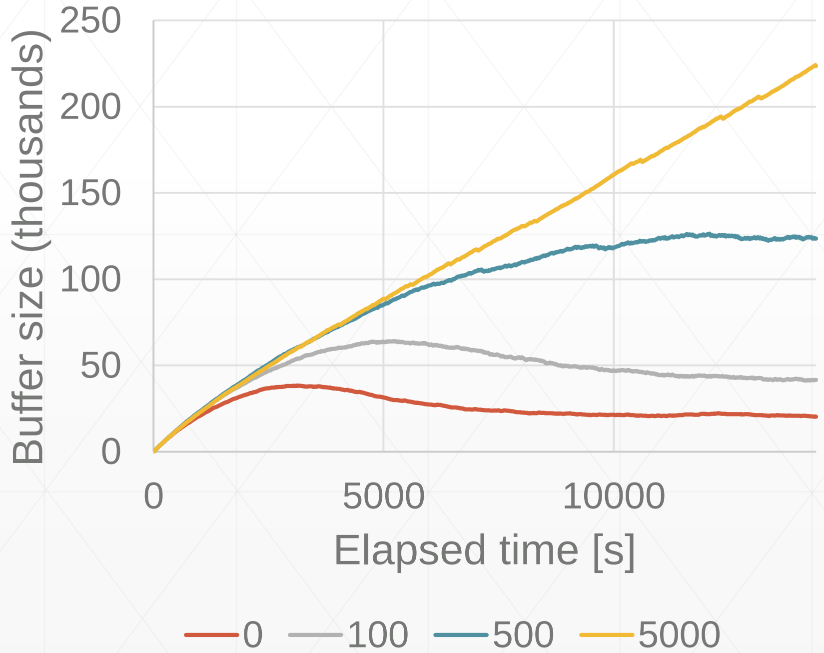
- Motivation
  - publish-subscribe application
    - requirement to obtain the latest data
    - e.g., 10% of the data is required to be processed with the delay of at most 1 minute
  - image annotation
    - requirement to search for latest images
    - 10% of images findable by keywords until 1 minute after their acquisition
- Goal: maximize expression w * |beforeDelayLimit| + |afterDelayLimit| for a given delay limit
  - |beforeDelayLimit| = set of query object processed until the given delay limit
  - |afterDelayLimit| = set of query object processed after the given delay limit
  - w = weight parameter
- Solution: modification of cluster ordering

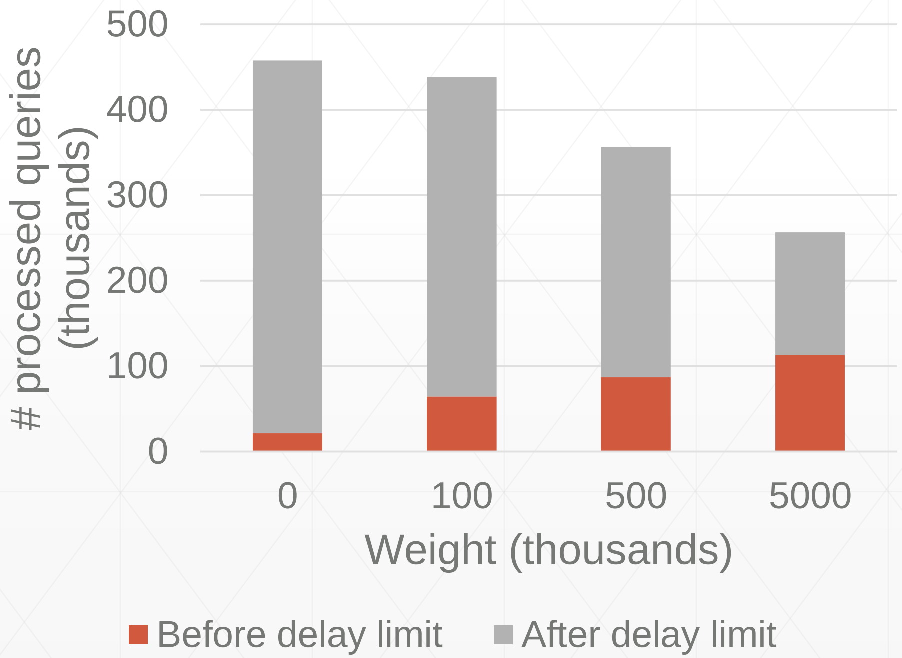
Throughput Delay Tradeoff Approach

- Original ordering: oldest cluster first
- Modification:
  - score for each cluster = a · |beforeLimitQueries| + b · oldestQueryAge
  - beforeLimitQueries: set of query objects younger than the delay limit
  - oldestQueryAge: age of the oldest query object in the cluster
  - a, b: weighting parameters
- Depth-first traversal of the tree of clusters: select a child with the highest score

Throughput Delay Tradeoff Approach Experiments

- 30 ms input frequency
- Delay limit: 1 minute
- Runtime: 4 hours
- Experiments with different „a“ weights (thousands in graphs)
- b weight = 1



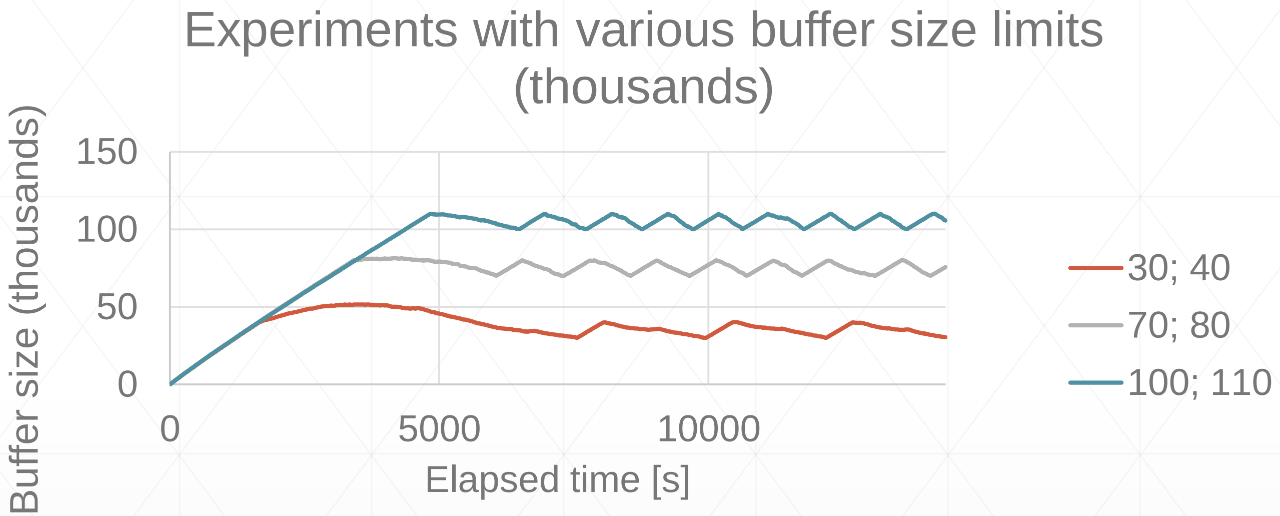


Throughput Delay Tradeoff Approach #2

- $a \cdot |beforeLimitQueries| + b \cdot oldestQueryAge$
- Switch between different strategies for cluster ordering, i.e., change the weights dynamically
  - throughput maximization: select cluster containing the oldest query object
    - $a = 0; b = 1$
  - maximization of low-delayed query objects: select a cluster containing the highest number of newest query objects
    - $a = 1; b = 0$
  - switch strategies based on buffer size limits
    - upper limit exceeded → maximize throughput
    - lower limit reached → focus on low delays

Throughput Delay Tradeoff Approach #2 Experiments

- 30 ms input frequency, different buffer size limits
- Delay limit (DL) = 1 minute



Lower limit	Upper limit	Queries before DL [%]
30,000	40,000	13
70,000	80,000	18
100,000	110,000	19

Results computed after 2nd switch

Summary

- Stream of similarity query objects
- Enhancing the throughput by query reordering and data partition caching
- Throughput delay tradeoff by modification of ordering strategies