Distributed Similarity Search Architectures

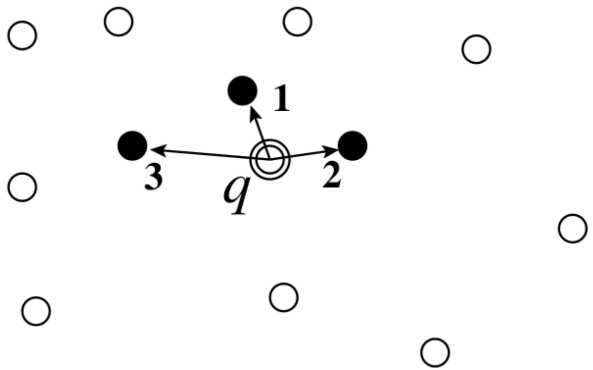
David Novak, Vladimír Míč, Pavel Zezula
DISA Lab

Metric-based Similarity

- generic similarity search
  - applicable to many domains

data modeled metric space *(D, δ)*, where *D* is a domain of objects and δ is a total distance function $\delta : D \times D \longrightarrow R^{+}_{0}$ satisfying postulates of identity, symmetry, triangle inequality

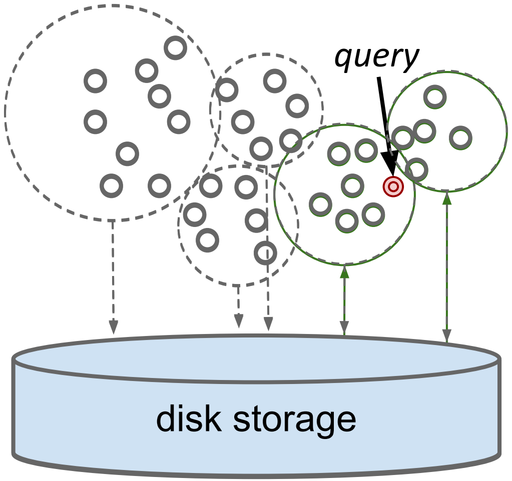
- search - query by example
- *K*-NN(*q*) query returns *K* objects *x* with the smallest $\delta(q,x)$



Similarity Indexing

- objective: organize the dataset $X \subseteq D$
    - so that similarity queries are processed efficiently

- data volumes can be large
- distance δ can be demanding



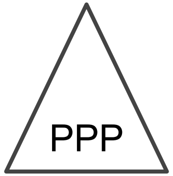
Objectives

- Distributed, horizontally scalable architecture
- ... for generic similarity search
- ... in Big data collections (hundreds of millions)
- ... single query efficiency
- … high query throughput

Outline

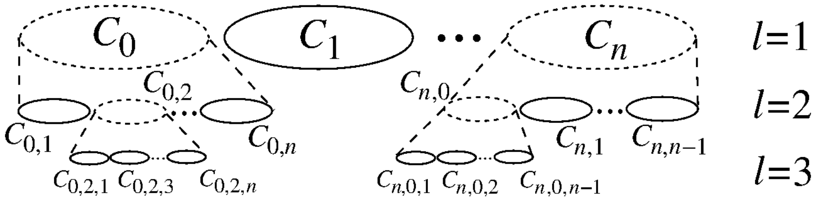
- *motivation*
- specific distributed systems
  - building blocks
  - our existing solutions (M-Chord, distributed M-Index)
  - other possible solutions
- analytical approach:
  - system model + cost model (just basic ideas)
- future work

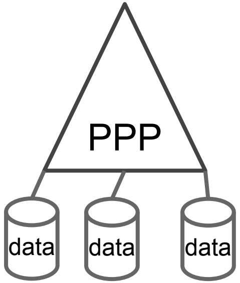
Building Blocks - Notation



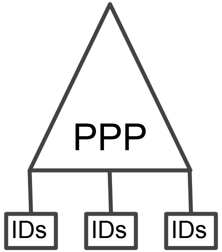
PPP-Tree (Pivot Permutation Prefix)

- recursive Voronoi tree



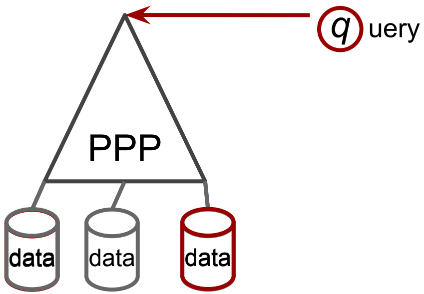


PPP-Tree with leaves pointing to buckets with data (disk or memory)



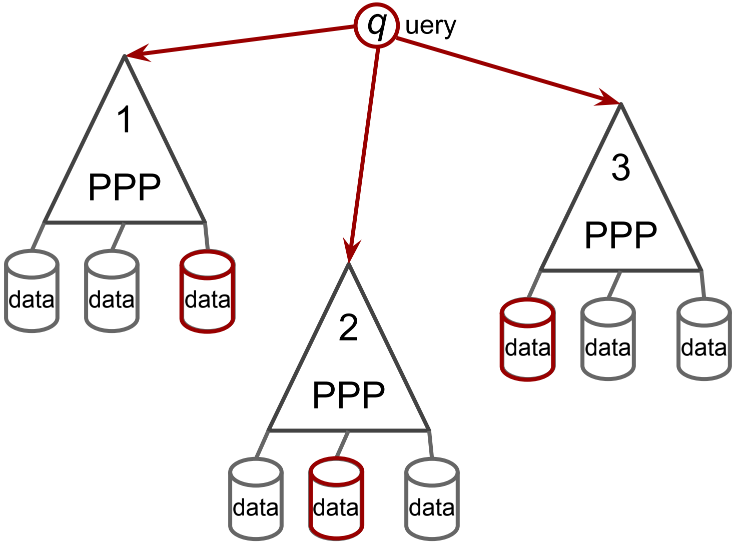
PPP-Tree with leaves storing only IDs of objects (typically memory)

M-Index



- simple PPP-Tree + memory or disk storage
- given an approximate query k-NN(q)
- most relevant buckets are (read from disk) and refined

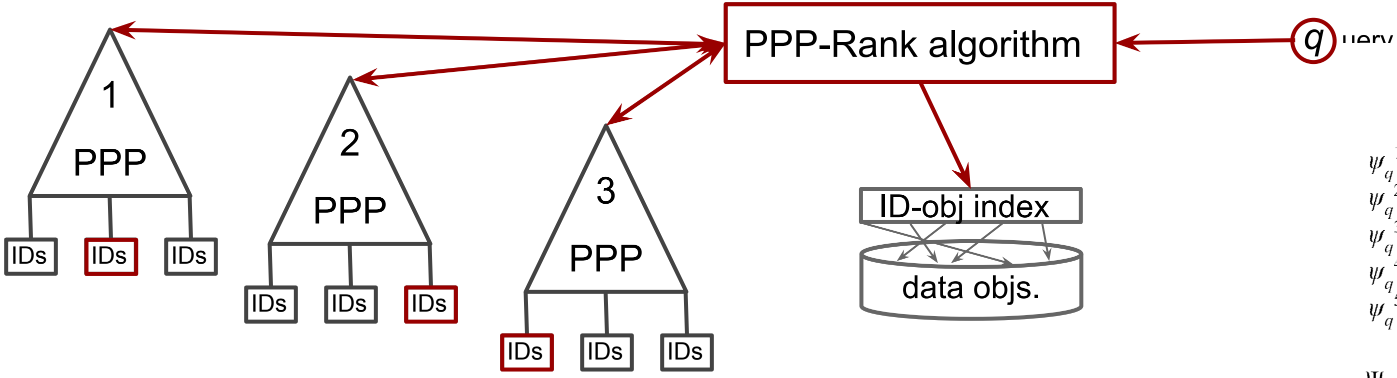
Multi M-Index



- λ independent PPP-Trees
- data in buckets
  - either in memory (shared)
  - or on disk (replicated)

- given query *k*-NN(*q*)
- most relevant buckets from each tree are accessed

PPP-Codes



$$\psi_q^{1}: \underbrace{\{\boldsymbol{x}\ y_1\ y_2\}}_{\text{objects with the rank '1'}}\ \underbrace{\{y_3\ y_4\ y_5\}}_{\text{rank '2'}}\ \underbrace{\{y_6\}}_{\text{rank '3'}} \ldots$$

$$\psi_q^{2}: \{y_3\ y_2\}\ \{y_1\ y_4\ y_6\ y_7\}\ \{\boldsymbol{x}\ y_8\} \ldots$$

$$\psi_q^{3}: \{\boldsymbol{x}\}\ \{y_3\ y_4\ y_5\}\ \{y_2\ y_6\} \ldots$$

$$\psi_q^{4}: \{y_1\ y_2\}\ \{y_3\ y_4\ y_5\}\ \{y_8\}\ \{y_6\} \ldots$$

$$\psi_q^{5}: \{y_1\ y_2\}\ \{y_4\ y_5\}\ \{y_3\}\ \{\boldsymbol{x}\ y_7\} \ldots$$

$$\Psi_{0.5}(q, x) = percentile_{0.5}\{1, 1, \mathbf{3}, 4, ?\} = 3$$

- λ independent PPP-Trees
  - IDs in leaf nodes (memory)
- ID-object storage
  - SSD disk

- given query *k*-NN(*q*)
  1. relevant leaves from each tree accessed (λ priority queues of IDs)
  2. PPP-Rank merges the ID queues
     - final candidate set is “very” small
  3. refine candidate set one-by-one

PPP-Code: Pros & Cons

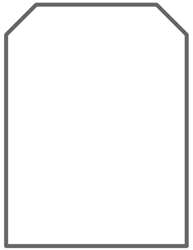
- Weak points
  - if dataset large, requires SSD (not feasible on HDD)
  - PPP-Trees with IDs take some memory
  - PPP-Rank algorithm takes some time

- Strong points
  - candidate set can be much smaller (2 orders of mgn.)
    - important for larger objects or expensive distance
  - data stored in ID-object store
    - a shared store for other indexes...

Distributed Indexes

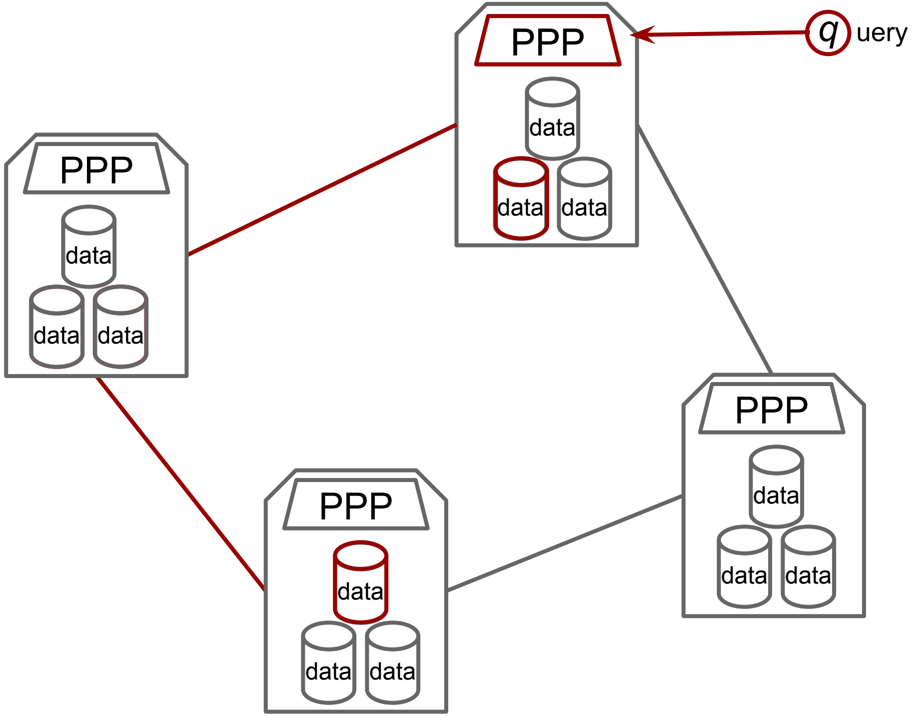
- M-Chord (2006)
- Distributed M-Index (2012)

- Future organizations



basic component of distributed system is a "node"

M-Chord



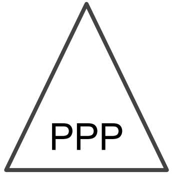
- basic component is "fixed-level" PPP mapping
  - static

- PPP mapping determines query-relevant buckets

- these buckets are accessed and refined on nodes

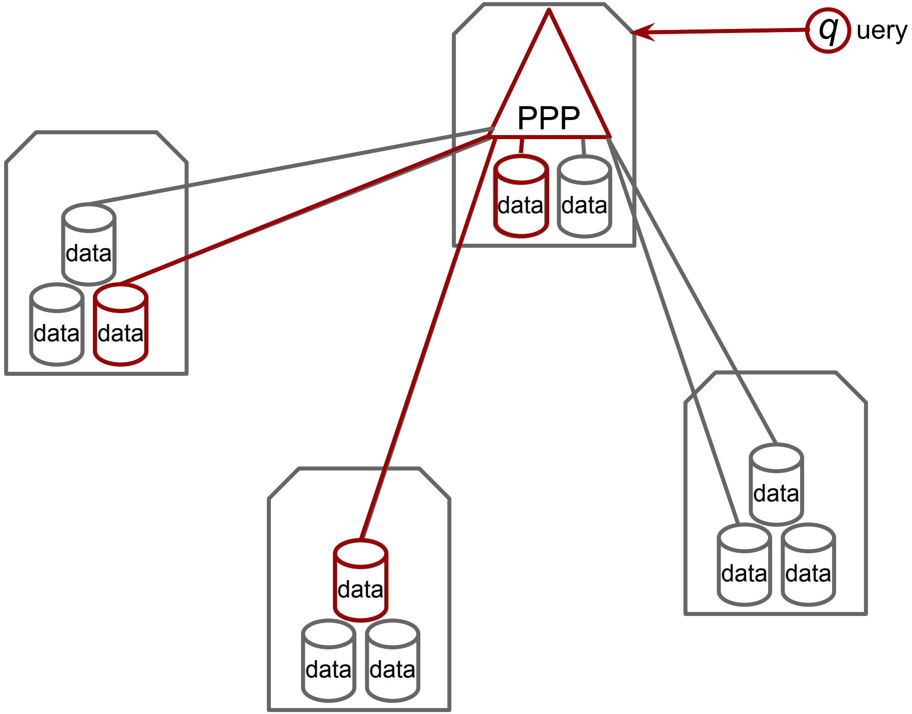
M-Chord: Pros & Cons

- The fixed (static) PPP is not that precise as dynamic PPP-Tree



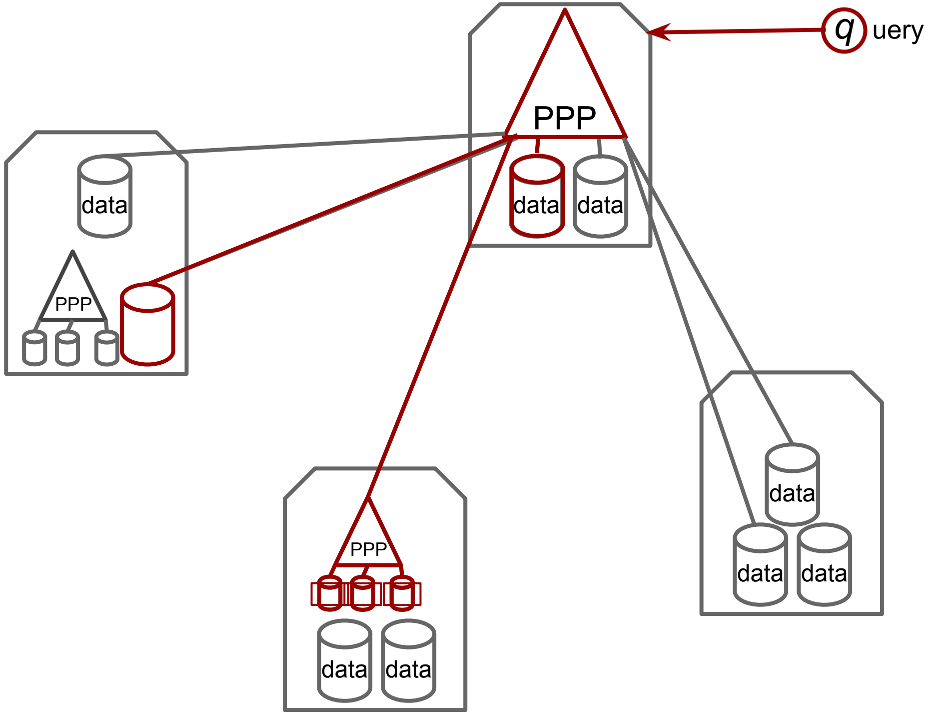
- ..but it allows easy replication

Distributed M-Index



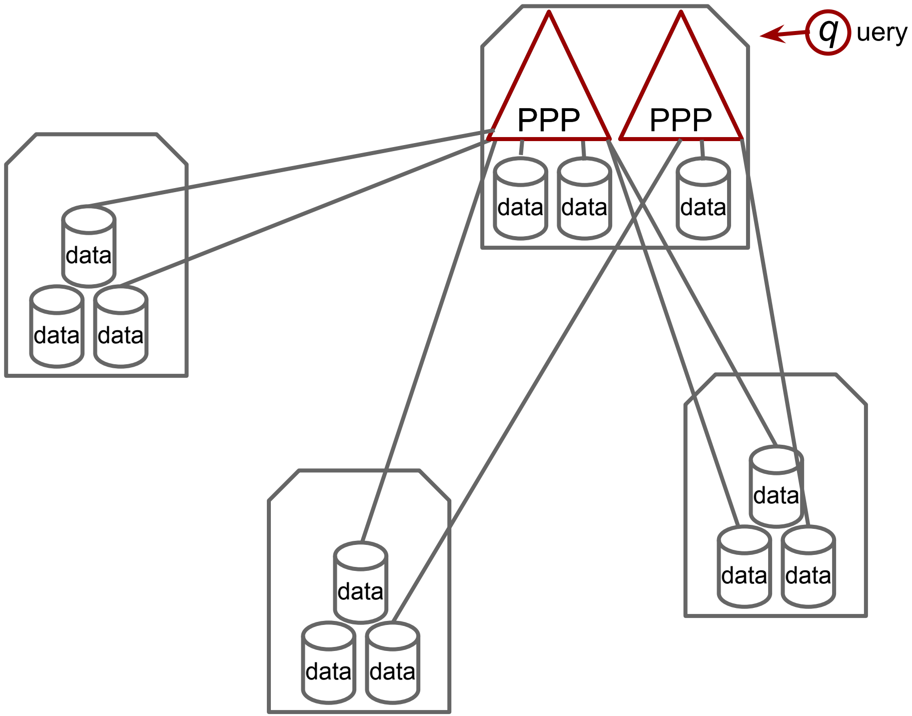
- Space partitioning by dynamic PPP-Tree
- PPP mapping determines query-relevant buckets
- these buckets are accessed and refined on nodes

Distributed M-Index (Local Indexes)



- Data buckets can be organized by local indexes

Distributed M-Index (Multiple PPPs)

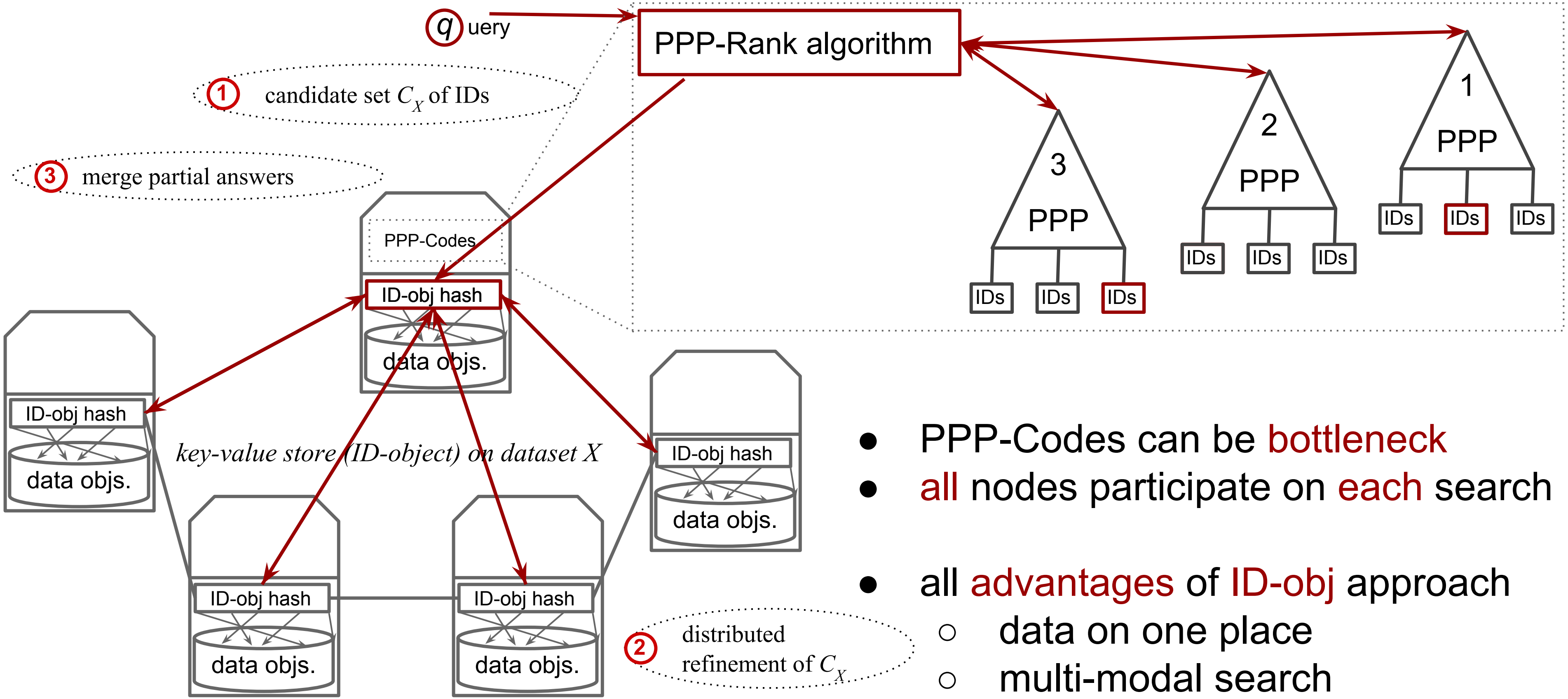


- Space partitioning by λ PPP-Trees
- Search in all PPP spaces
- Data replicated λ-times

M-Chord + Distribute M-Index

- Relatively large candidate set
  - but navigation on the bucket level - few messages
- Data distributed by the similarity space
  - data retrieved by sequential reads (HDD)
    - ...but difficult to build secondary indexes on the same data
    - e.g. ID-object index

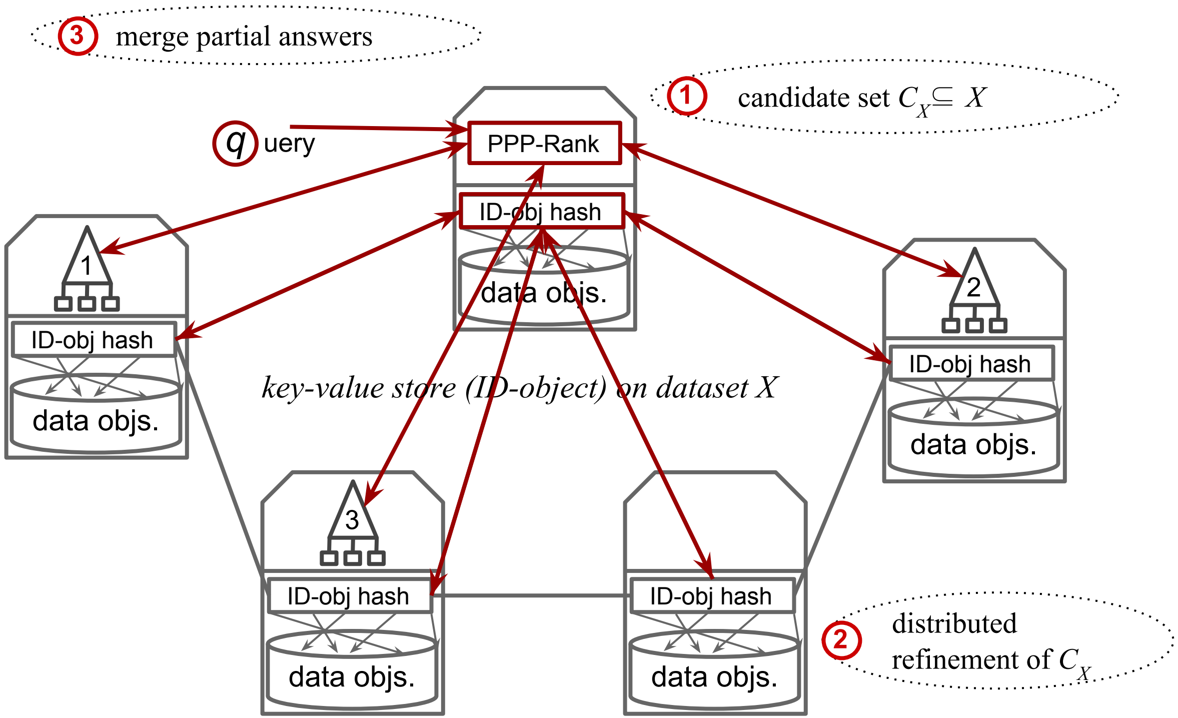
Distributed PPP-Codes: Variant 1



- PPP-Codes can be bottleneck
- all nodes participate on each search

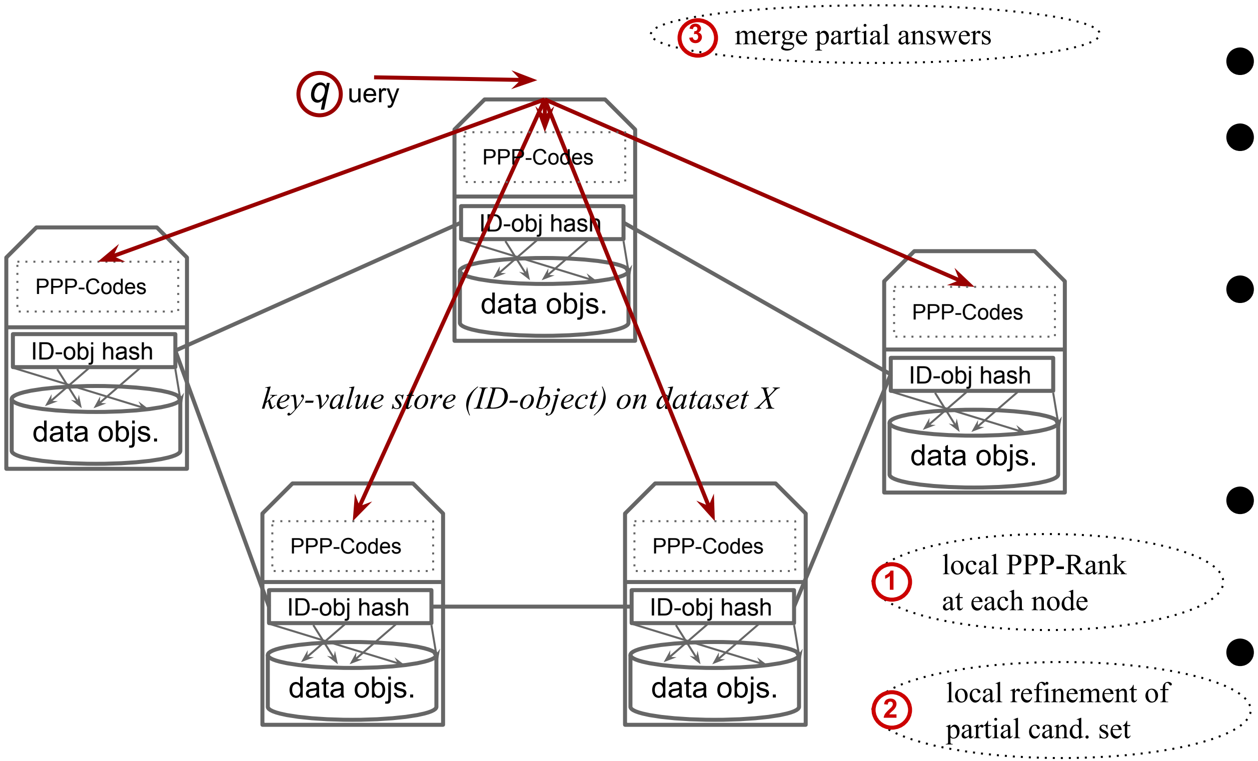
- all advantages of ID-obj approach
  - data on one place
  - multi-modal search

Distributed PPP-Codes: Variant 2



- PPP-Codes index is somewhat distributed
  - "no" bottleneck
- but complex communication during PPP-Ranking

PPP-Codes on Local Data



- no central point
- all *m* nodes participate
- uniform load distribution

- replication of each partition (3-times)
- query: access btw. *m*/3 to *m* nodes

System Model

1. What global metric approach is used?
    - fixed/dynamic Voronoi, PPP-Codes, other (sketches)
    - result: data *partitions* (buckets, single objs., sketches)
2. How are the data partitions distributed?
    - using the metric partitioning, independent hash, …
3. Other questions:
    - replication of partitions
    - local indexes on the level of partitions or nodes
    - communication among nodes (log *n* or direct)

Cost Model

Analytical evaluation of:

- time of a one query processing
- query throughput
    - number of queries per second

1. Derive formulas for different system settings
2. Simulate costs for various parameters

Cost Model: Variables

- size of dataset: *n* objects
- number of nodes: *m*
- *time(d)* of d(q,x) comp. [ms], size of each objects [B]
- I/O speed [B/s], network latency [ms] & bandwidth [B/s]

For each approach (get from papers or measure)

- # of partitions
- size of candidate set |*C(q)*| for given query recall
- time of C(q) generation
- distribution of partitions and candidate parts. to nodes

Cost Model: Query Processing

1. processing on the coordinating node:
   #pivots * *time(d)* + cand_generation_time(|C(q)|)   [ms]
2. communication:
   latency + (message_size / bandwidth) [ms]
3. slowest node processing:
   #objects_to_process = max_cand_frac * |C(q)|
   - read the data:
     #objects_to_process * size(x) / I/O_speed   [ms]
   - refine the data:
     #objects_to_process * time(d)   [ms]
4. communication back: latency + (0 / bandwidth)
5. merge of the answers and return

Index_type	Mchord	mindex_fixed	mindex_fixed metric	mindex_dynamic	mindex_dynamic metric	mindex_multi	mindex_multi metric	ppp_centralized
Počet pivotů (p)	100	100	100	100	100	200	200	400
Počet vrstev dělení (L)	1	1	1	1	1	4	4	4
Data_partitioning (hash/metric)	hash	hash	metric	hash	metric	hash	metric	hash
cand_size	80,000	80,000	80,000	50,000	50,000	80,000	80,000	5,000
cand_generation [ms]	0	0	0	5	5	10	10	200
partition_count	100	1,000	1,000	10,000	10,000	10,000	10,000	1,000,000
max_cand_frac	1	1	1	1	1	0.2	0	1
Hodnoty počítané stejným způsobem pro každý index:								
# accessed_partitions	8	80	80	500	500	800	800	5,000
message_count	1	1	POWER LAW	1	POWER LAW	1	POWER LAW	1
avg_message_size	32	320	#VALUE!	2,000	#VALUE!	3,200	#VALUE!	20,000
Výsledné časy [ms]								
Index build	1,000,000	1,000,000	1,000,000	1,000,000	1,000,000	2,000,000	2,000,000	4,000,000
Query “parallel cost”	992	992	#VALUE!	626	#VALUE!	211	#VALUE!	267
processing on the coordinating node	1	1	1	6	6	12	12	204
message sending	0	0	#VALUE!	0	#VALUE!	0	#VALUE!	1
slowest node read	191	191	191	119	119	38	38	12
slowest node refine	800	800	800	500	500	160	160	50
message sending	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
merging	0	0	0	0	0	0	0	0
Query “overall cost”	992	992	#VALUE!	626	#VALUE!	1,003	#VALUE!	267
processing on the coordinating node	1	1	1	6	6	12	12	204
message sending	0	0	#VALUE!	0	#VALUE!	0	#VALUE!	1
nodes read	191	191	191	119	119	191	191	12
nodes refinement	800	800	800	500	500	800	800	50
message sending	0	0	#VALUE!	0	#VALUE!	0	#VALUE!	0
merging	0	0	0	0	0	0	0	0

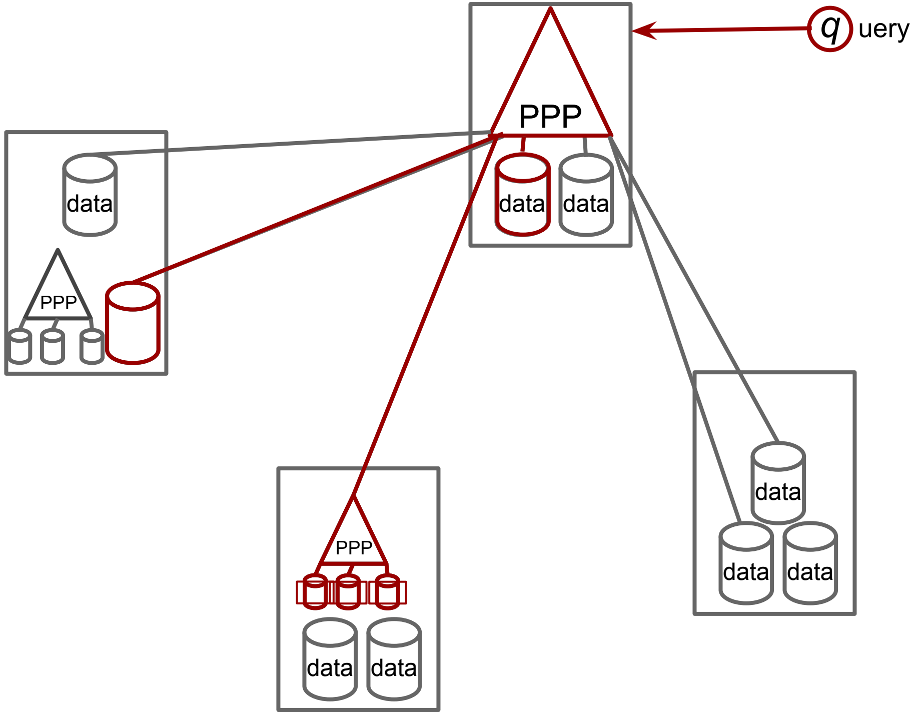
Velikosti kandidátních množin Recall	Mchord	mindex_fixed	mindex_fixed metric	mindex_dynamic	mindex_dynamic metric	mindex_multi	mindex_multi metric	ppp_centralized
50%	5.00%	5.00%	5.00%	3.50%	3.50%	5.00%	5.00%	0.18%
60%	6.00%	6.00%	6.00%	4.00%	4.00%	6.00%	6.00%	0.20%
70%	7.00%	7.00%	7.00%	4.50%	4.50%	7.00%	7.00%	0.30%
80%	8.00%	8.00%	8.00%	5.00%	5.00%	8.00%	8.00%	0.50%
90%	9.00%	9.00%	9.00%	5.50%	5.50%	9.00%	9.00%	0.60%
95%	15.00%	15.00%	15.00%	7.50%	7.50%	12.00%	12.00%	1.50%
99%	20.00%	20.00%	20.00%	10.00%	10.00%	15.00%	15.00%	2.00%

Fixní parametry	Recall	Velikost indexované množiny (n)	Počet uzlů (m)
Hodnoty	0.8	1,000,000	
Definiční obor	{0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99}	setN	{1, 5, 10, 20

Fixní parametry	time(d) [ms]	latency [ms]	Bandwidth [B/s]
Hodnoty	0.01	0.2	
Definiční obor	setR+	setR+	

Thank you for your attention

Distributed M-Index (Local Indexes)



- Data buckets can be organized by local indexes

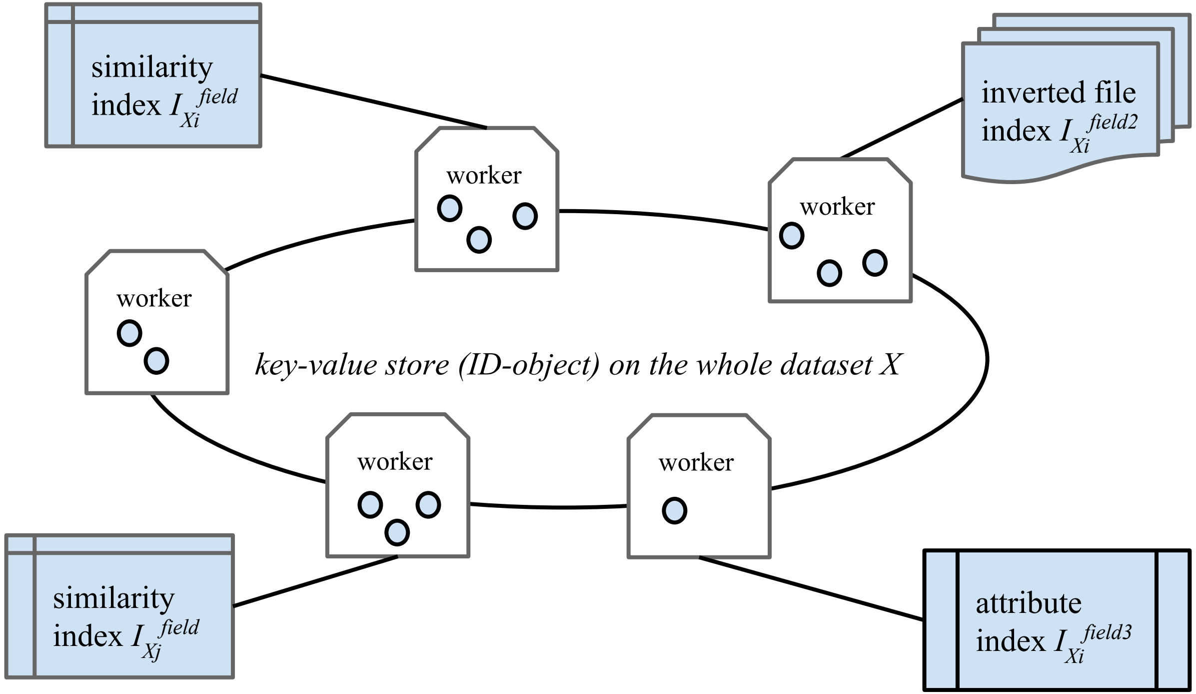
Advantages of ID-object store

- ID-object queries (w/o another database)
    - all data in one places (good for consistency)
- efficient access by multiple modalities
    - data not partitioned by a single similarity modality
    - enables indexes on other attributes/modalities
    - final ranking by combination of modalities + filtering
- multiple collections use common ID-obj. store

M-Chord + Distribute M-Index

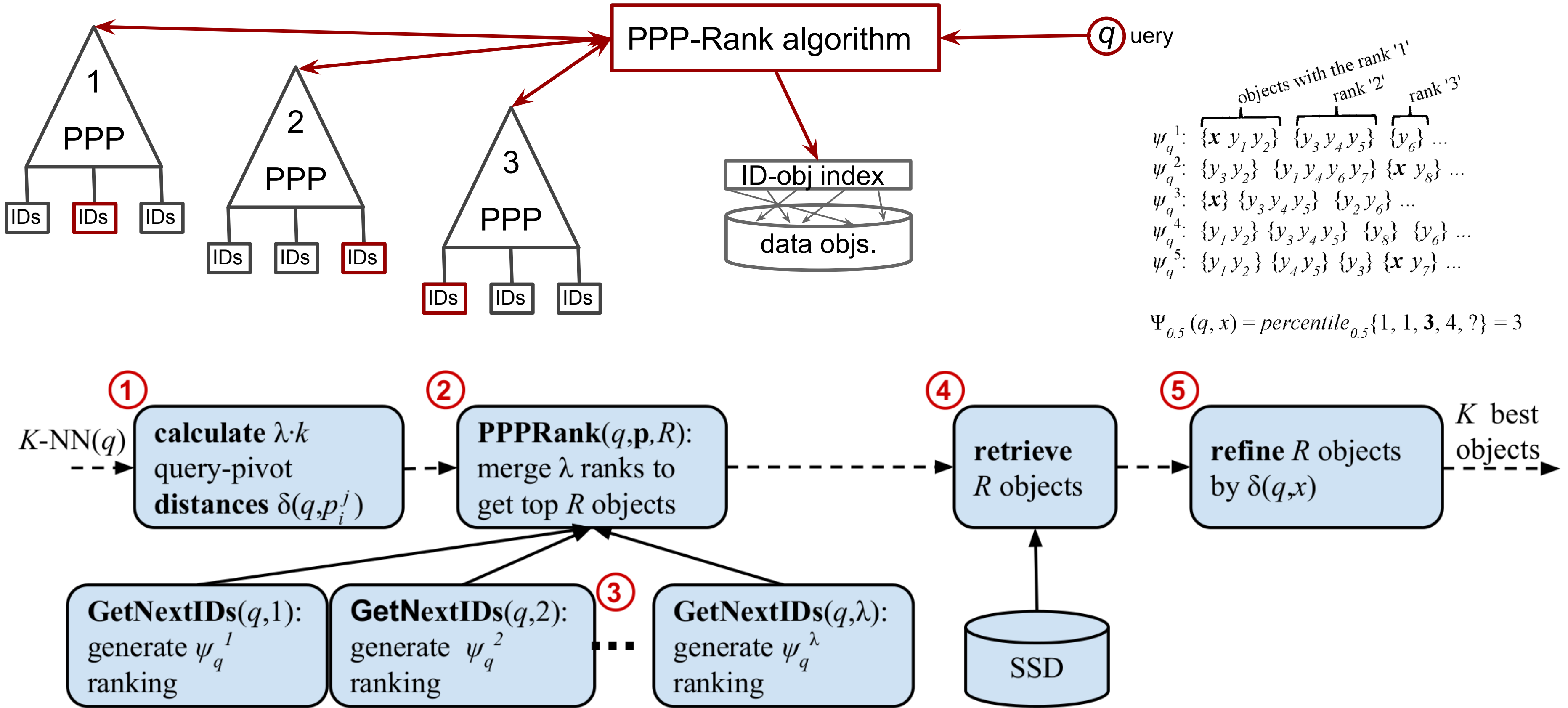
- Relatively large candidate set
    - but navigation on the bucket level - few messages
- Data distributed by the similarity space
    - data retrieved by sequential reads (HDD)
        - ...but difficult to build secondary indexes
    - e.g. ID-object index could be built, but:
        - global part of the index + local on each bucket
        - as index grows, data is redistributed
            - data objects moved to other buckets
        - another similarity index almost impossible

Distributed PPP-Codes: Other Indexes

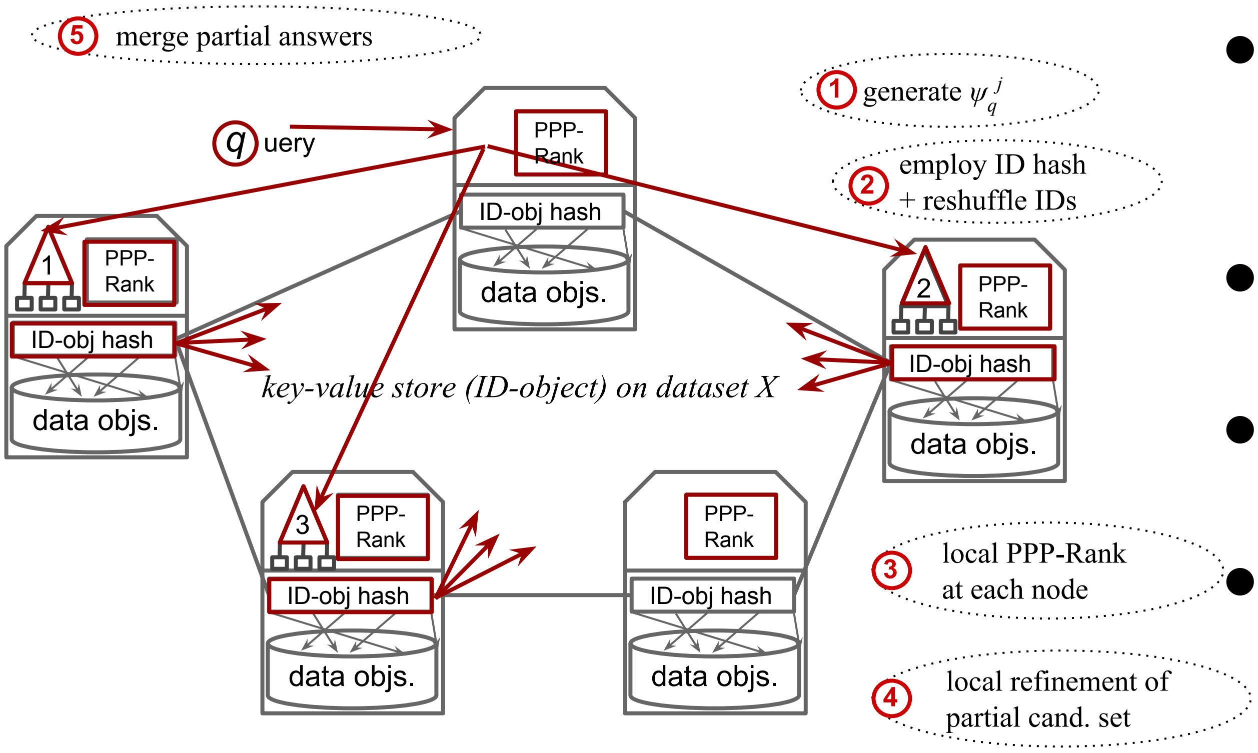


- other indexes:
  - multi-modal search: combination, filtering, re-ranking

PPP-Rank: the Details



Distributed PPP-Codes: Variant 3



- The PPP-Ranking is fully distributed
  - to all nodes
- Communication is "one-way"
- PPP-Trees can be replicated
- Size of ψ_q^j must be set a priori