Efficient Processing of Hamming-Distance-Based Similarity Search Queries over MapReduce

Mingjie Tang* Yongyang Yu* Walid G. Aref*
Qutaibah Marwan Malluhi^ Mourad Ouzzani~

*Purdue University ^Qatar University
~Qatar Computing Research Institute

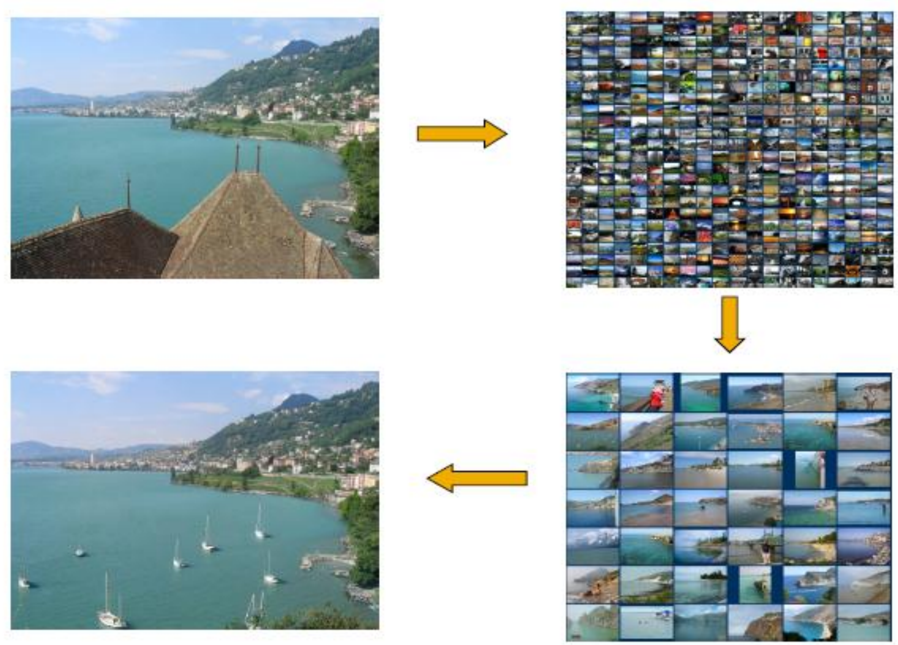
EDBT 2015

Why is it Important?

- Content-based similarity search
  - Image, Text, Video, Fingerprint
  - Similarity search for the transformed high dimensional vector
- **Query:**

  Find similar Images

  to a query image from an image collection (J. Leskovec et al. 2013)



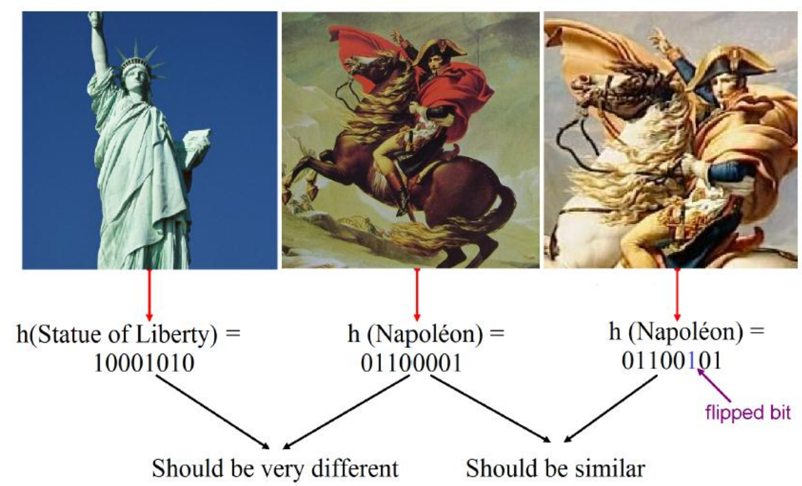
Some Preliminaries (1/3)

- Similarity Preserving Hashing is widely used for approximate near neighbor(NN) search
  - ✓ Data independent similarity hashing (LSH)
  - ✓ **Data-dependent similarity hashing**
- Core operation of **data-dependent similarity hashing** is Hamming distance range query

  (*Wujun et al. 2013*)

Step1: Hashing data to binary

Step2: Hamming-select



Preliminaries (2/3)

- Hamming Distance:
  - between two strings of equal length is the number of positions at which the corresponding symbols are different
- Example
  - String: “abcdd” and “cbcaa” is 3
  - Binary codes: **1011101** and **1001001** is 2

Preliminaries (3/3)

- Hamming-select
  - Return tuples from a dataset, where the Hamming distance to a point query is not bigger than a predefined threshold **H**
  - Given threshold **H**=3 and Q=101100010
  - Hamming-select
  - (Q, S)={t0, t3, t4, t6}
- Hamming-join
  - Threshold **H**=3 Hamming-join(R,S)=

{(r0, t0), (r0, t3), (r0, t4), (r0, t6)}

{(r1, t0), (r1, t3), (r1, t4), (r1, t6)}

{(r2, t3)}.

(a) Table S

tuple	binary U
t_0	001 001 010
t_1	001 011 101
t_2	011 001 100
t_3	101 001 010
t_4	101 110 110
t_5	101 011 101
t_6	101 101 010
t_7	111 001 100

(b) Table R

tuple	binary U
r_0	101 100 010
r_1	101 010 010
r_2	110 000 010

Related Work

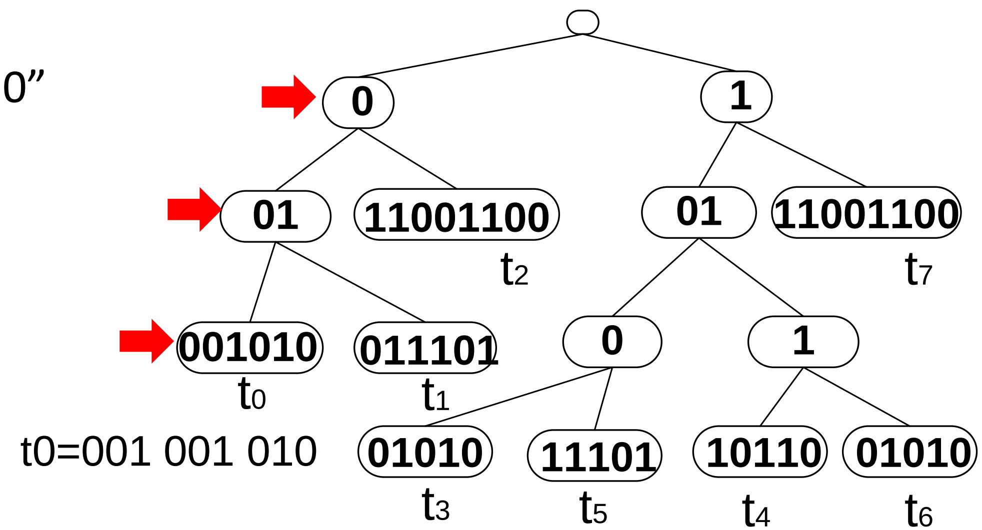
- Hamming distance range query processing
- Only work for small H
  - Yao et al. 1974
- High overhead of memory usage to maintain several copies of data
  - Mutli-HashTable (Manku et al. 2007), Hengine (Liu et al. 2012), HmSearch (Zhang et al. 2013)
- Only work for small datasets
  - Hengine(Liu et al. 2012), HmSearch (Zhang et al. 2013)

Challenges

- Efficient Hamming-select
  - new index to support efficient Hamming-select
    - efficient data update
    - small space footprint
- Efficient Hamming-join
  - efficient parallel algorithm for Hamming-join over two big tables
    - handle load balancing
    - guarantee lower data shuffle cost

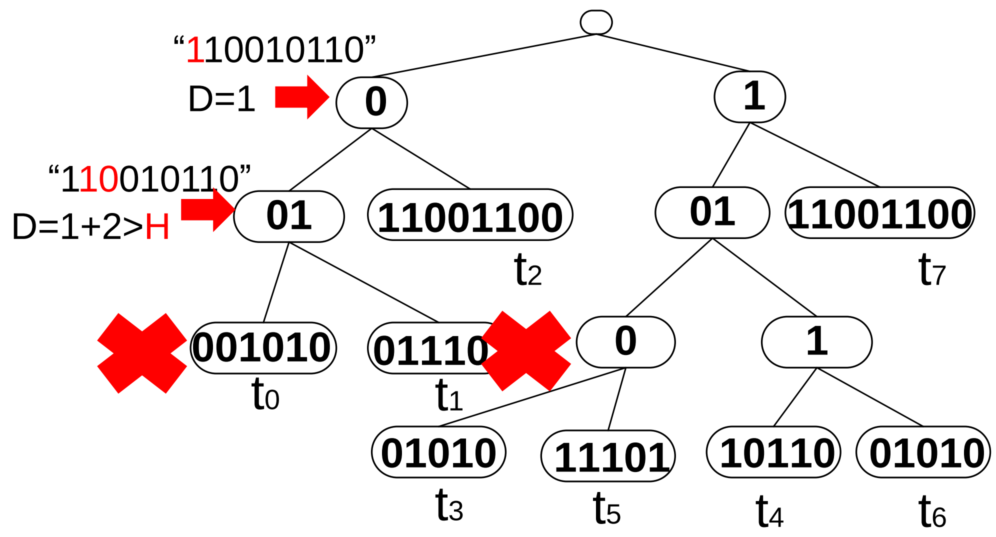
Optimization Algorithm for Hamming-Select

- (A) Build Radix-Tree via prefix properties of data
  - Internal nodes store prefix
  - Leaf node stores tuple ID
  - Example:
    - Given Tuple t0="001 001 010"
    - Prefix patterns of t0
    - "0" "01" "001010"



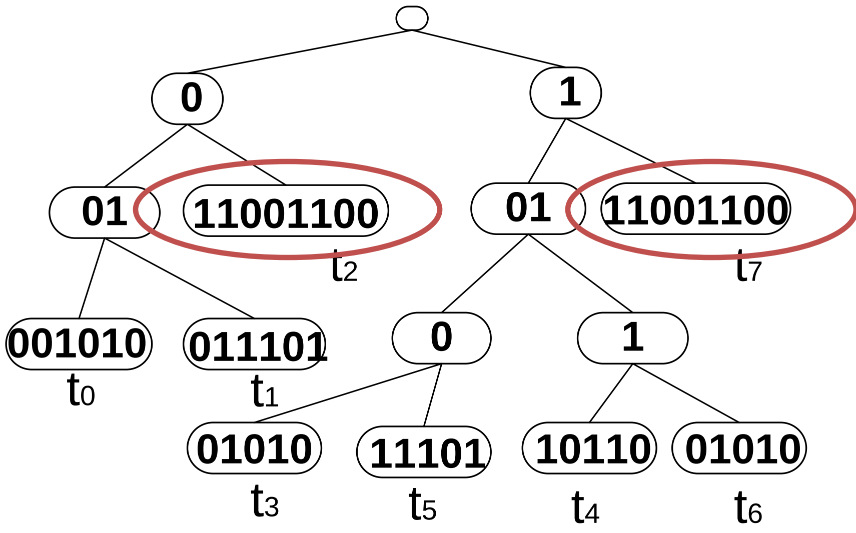
Optimization Algorithm for Hamming-Select

- (A) Query Radix-Tree for Hamming-Select
  - Search from root to leaf
  - Example:
    - Given Query="110010110" and Threshold H="2"
    - Tuple t0 and t1 are discarded
    - Stop in upper level of Index



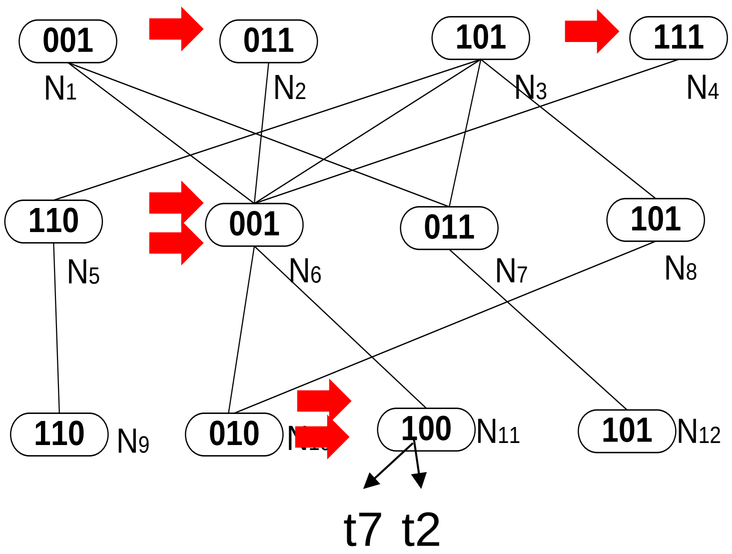
Optimization Algorithm for Hamming-Select

- (A) Query Radix-Tree for Hamming-Select
  - Search from root to leaf
  - Prefix Sensitive
  - t2: "0**11001100**"
  - t7: "1**11001100**"



Optimization Algorithm for Hamming-Select

- (B) Static-HA-Index
- Motivation:
  - Consider Tuple t2 and t7
  - “011 001 100” vs “111 001 100”
  - Reduce the redundant computation?
  - Segmentation of binary code
  - For example:
    - t2 is segmented into “011 001 100”
    - t7 is segmented into “111 001 100”
  - Hamming-search from first level to bottom level



Optimization Algorithm for Hamming-Select

- (C) Gray-code-based ordering, Hamming distance clustering (Ral et al. 1991) for Dynamic HA-Index
- Observation:

(a) Table S

tuple	binary U
t_0	001 001 010
t_1	001 011 101
t_2	011 001 100
t_3	101 001 010
t_4	101 110 110
t_5	101 011 101
t_6	101 101 010
t_7	111 001 100

t0
t1
t2
t7
t4
t6
t3
t5

t2: 011 001 100
t7: 111 001 100

Hamming (t2, t7)=1

Build Dynamic HA-Index for Hamming-Select

– Step 1: Sort via Gray code order

Before sorting:

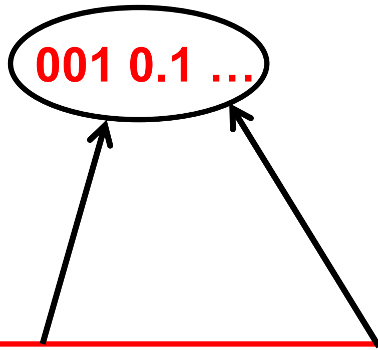
t0="001 001 010", t1="001 011 101", t2="011 001 100", t3="101 001 010",
t4="101 110 110", t5="101 011 101", t6="101 101 010", t7="111 001 100",

After sorting:

t0="001 001 010", t1="001 011 101", t2="011 001 100", t7="111 001 100",
t4="101 110 110", t6="101 101 010", t3="101 001 010", t5="101 011 101"

Build Dynamic HA-Index for Hamming-Select

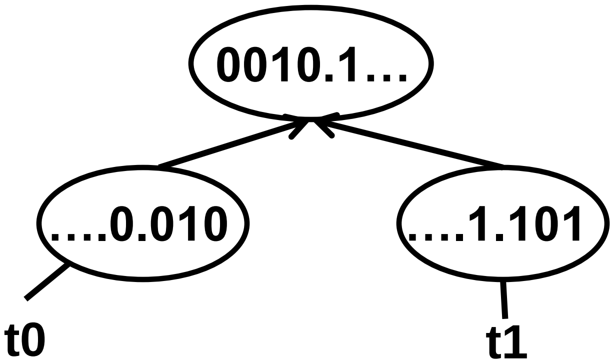
– Step 2: Extract common sub-sequence from binary in the same window



t0="**001 0**0**1** 010", t1="**001 0**1**1** 101", t2="011 001 100", t7="111 001 100",
t4="101 110 110", t6="101 101 010", t3="101 001 010", t5="101 011 101"

Build Dynamic HA-Index for Hamming-Select

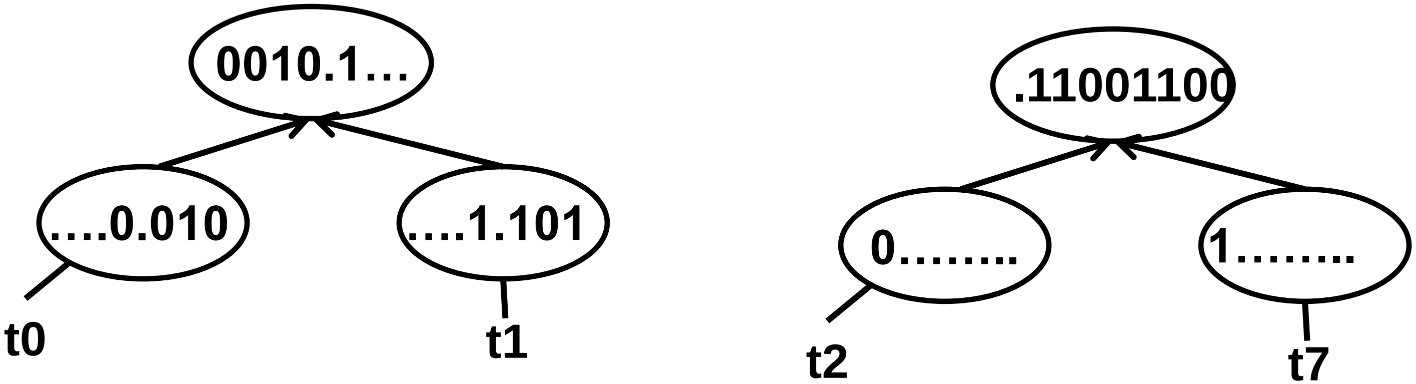
– Step 2: Extract common sub-sequence from binary in the same window



t0="… .0. 010", t1="… .1. 101", t2="011 001 100", t7="111 001 100", t4="101 110 110", t6="101 101 010", t3="101 001 010", t5="101 011 101"

Build Dynamic HA-Index for Hamming-Select

– Step 2: Extract common sub-sequence from binary in the same window

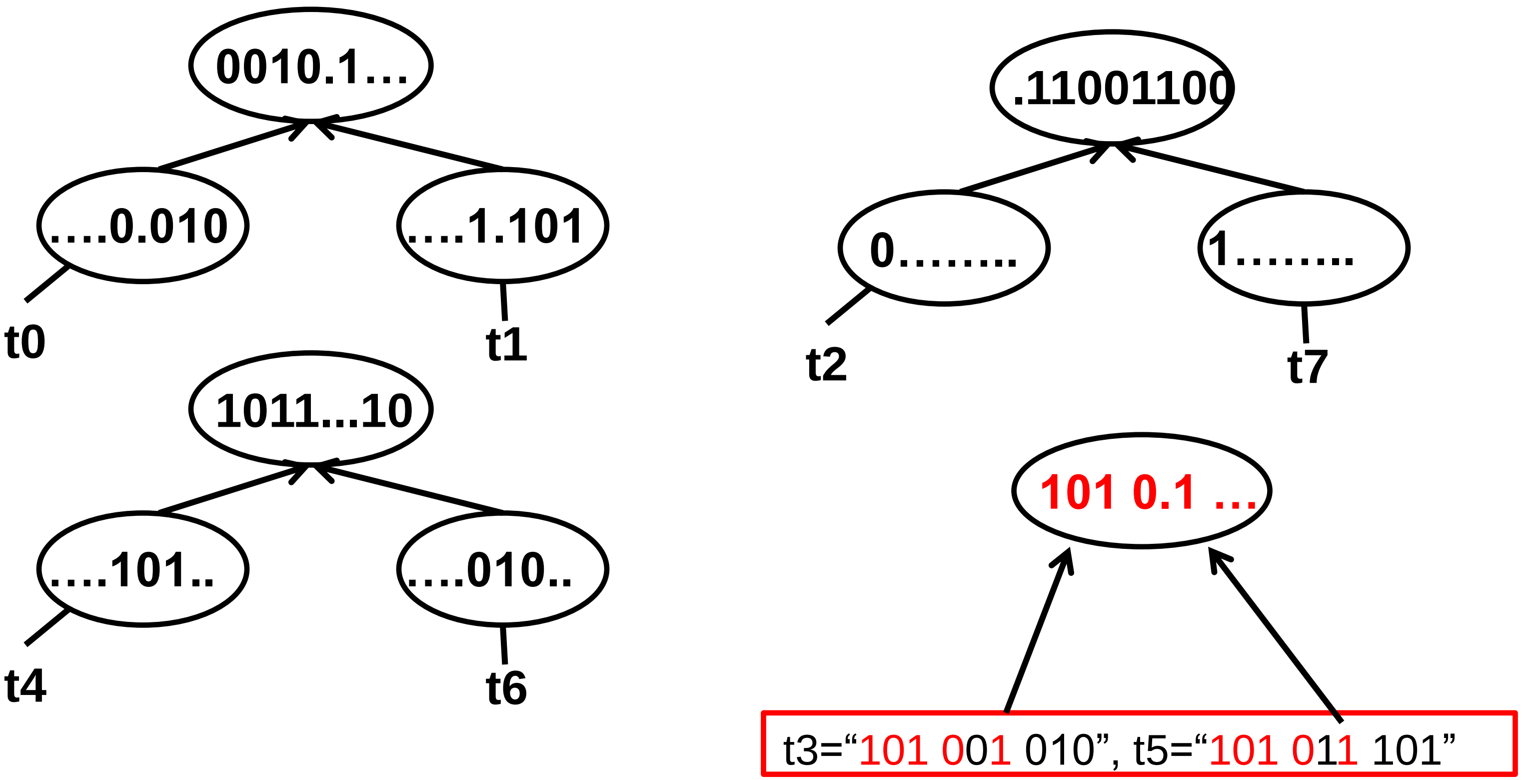


t2="011 001 100", t7="111 001 100",

t4="101 110 110", t6="101 101 010", t3="101 001 010", t5="101 011 101"

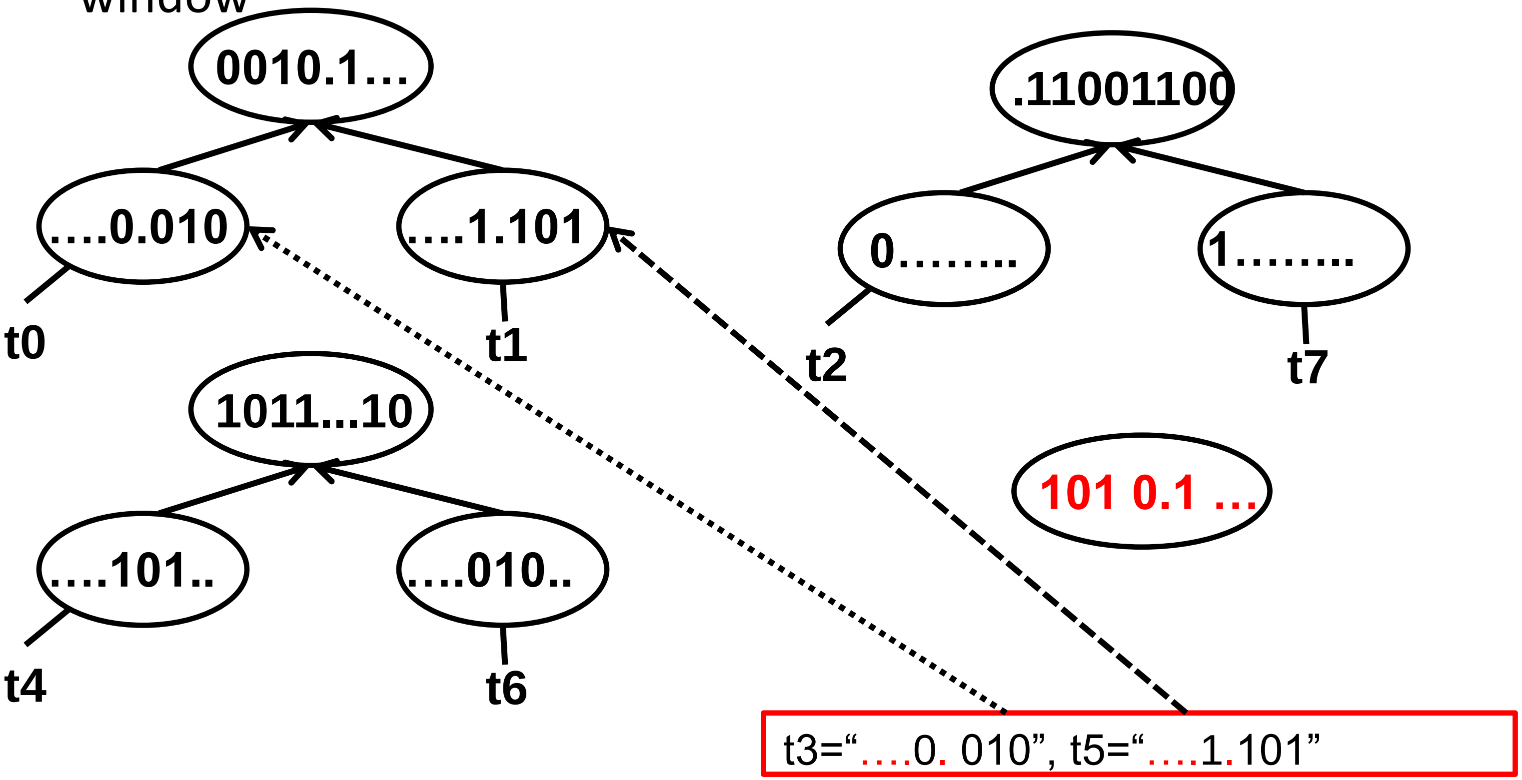
Build Dynamic HA-Index for Hamming-Select

– Step 2: Extract common sub-sequence from binary in the same window



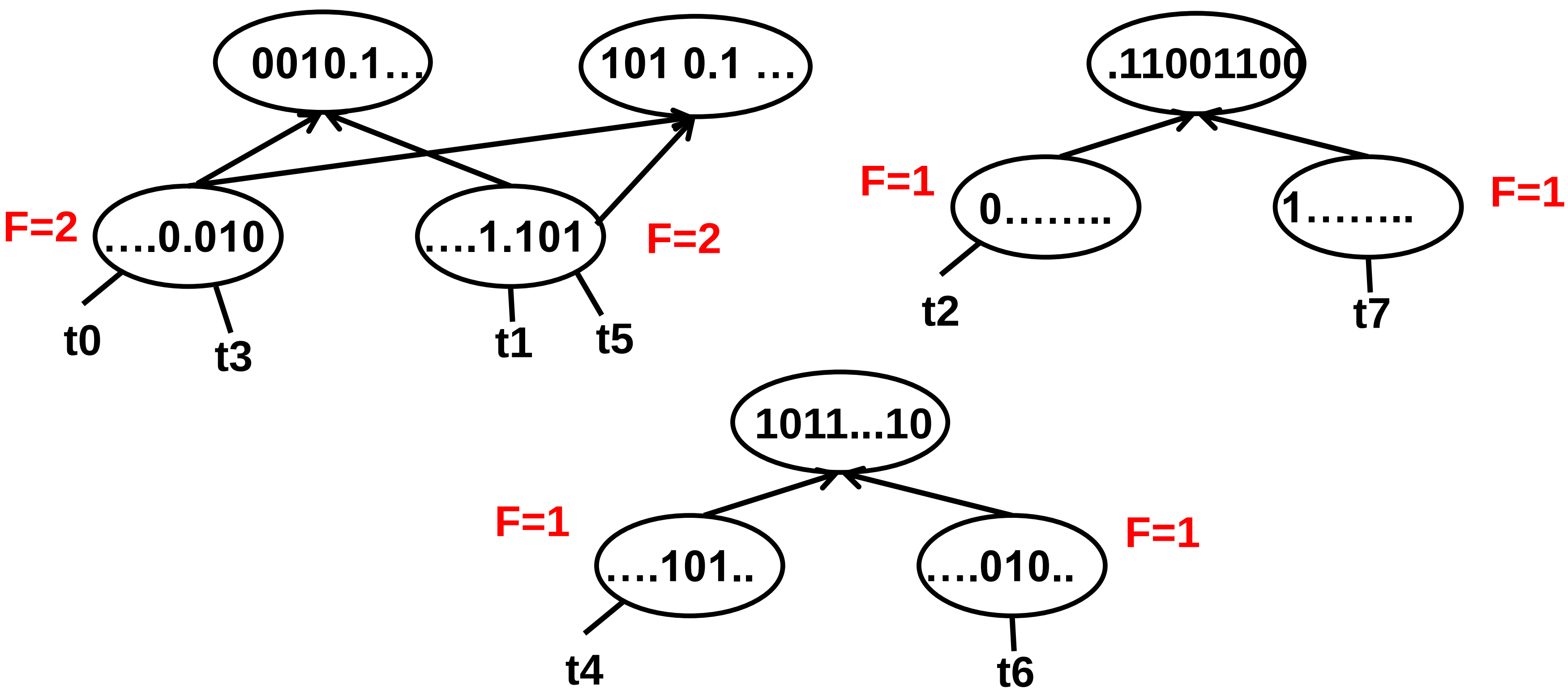
Build Dynamic HA-Index for Hamming-Select

– Step 2: Extract common sub-sequence from binary in the same window



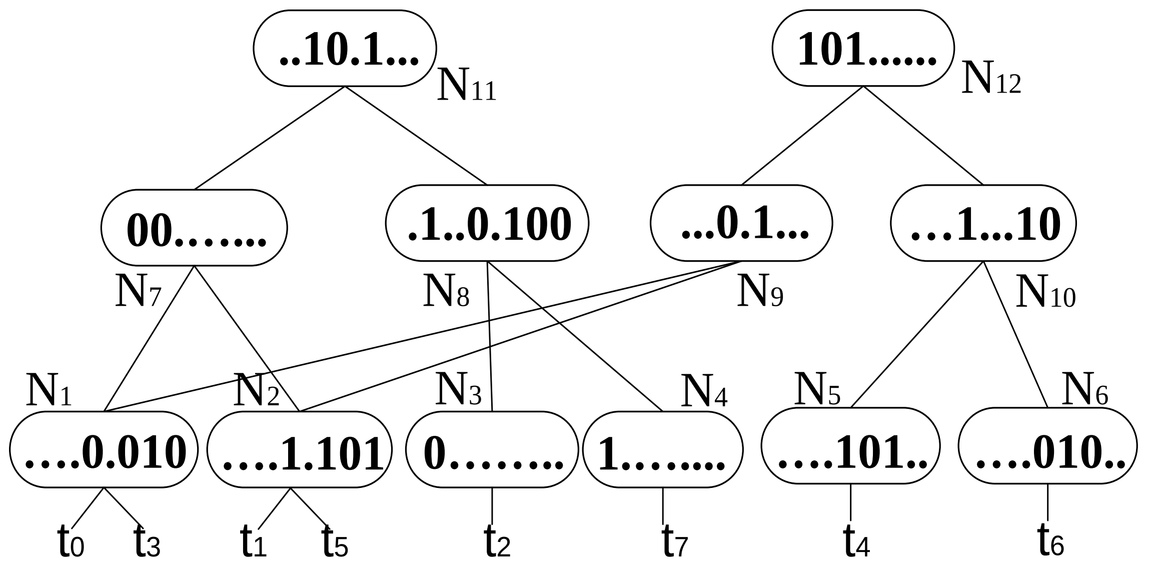
Build Dynamic HA-Index for Hamming-Select

– Step 2: Extract common sub-sequence from binary in the same window



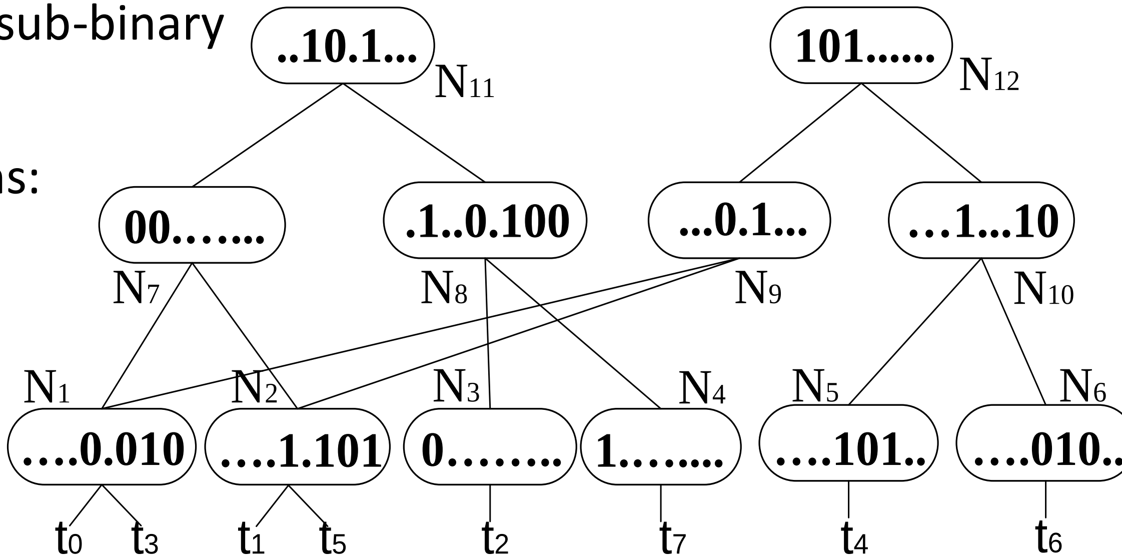
Optimization Algorithm for Hamming-Select

- Build Dynamic HA-Index for Hamming-Select
  - Step 3: Continue the same process in each level



Optimization Algorithm for Hamming-Select

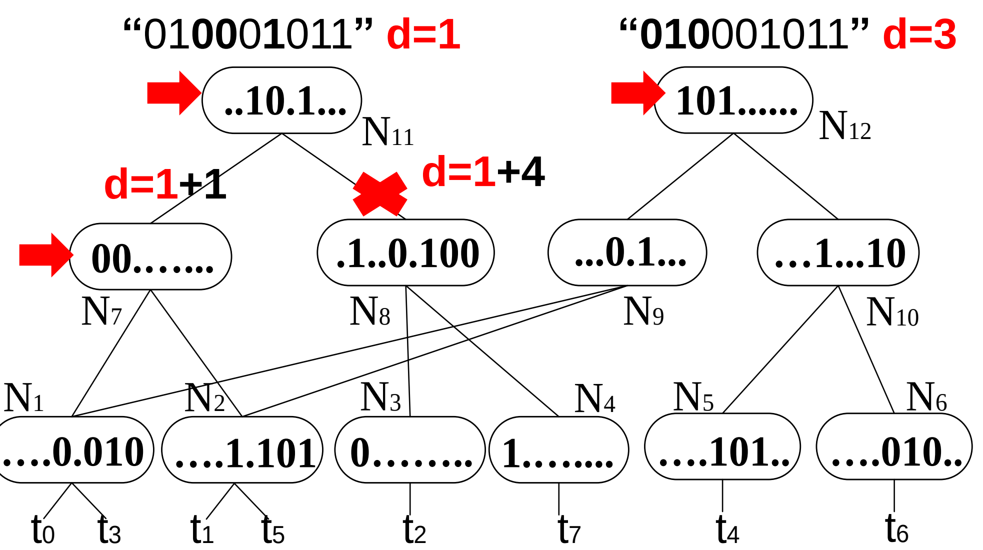
- (C) Summary of Dynamic HA-Index
  - Build index via Gray code ordering
  - Internal node: common sub-binary
  - Leaf node: tuple id
  - Support Index Operations:
    - (A) **Build**
    - (B) Delete
    - (C) Insert
    - (D) Hamming-select



HA-Index based

- Hamming-Select
  - Searching from top to bottom (Breadth-First-Search or Depth-First-Search)
  - Given Binary ="010001011" ,

  and Hamming-distance threshold=3
  - Running Example

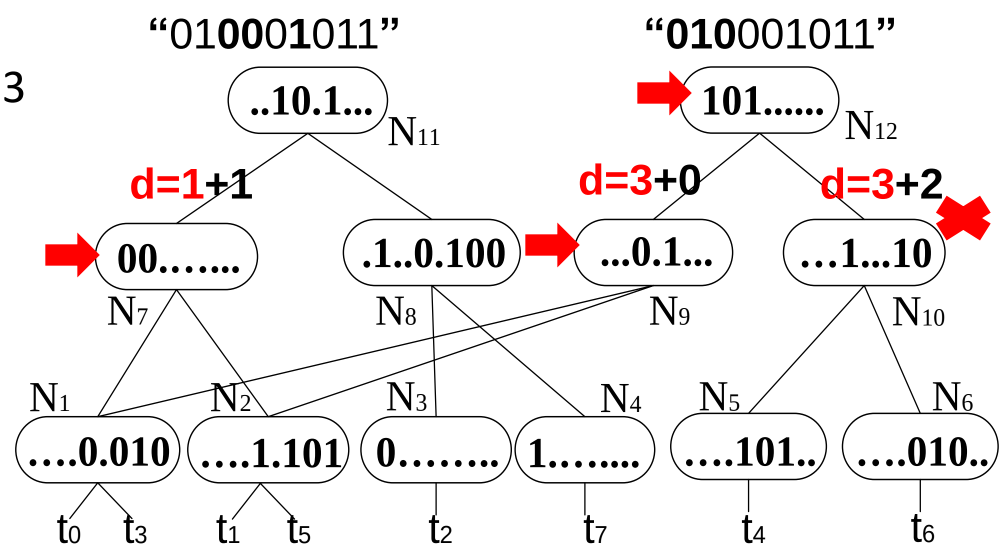


Queue	Qualified tuples ret
N_{11}, N_{12}	$\emptyset$
$N_{12}, [N_7, N_{11}]$	$\emptyset$
$[N_7, N_{11}], [N_9, N_{12}]$	$\emptyset$
$[N_9, N_{12}]$	t_0
$\emptyset$	t_0

HA-Index based

- Hamming-Select
  - Searching from top to bottom (Breadth-First-Search or Depth-First-Search)
  - Given Binary T="010001011" ,

  and Hamming-distance threshold=3
  - Running Example



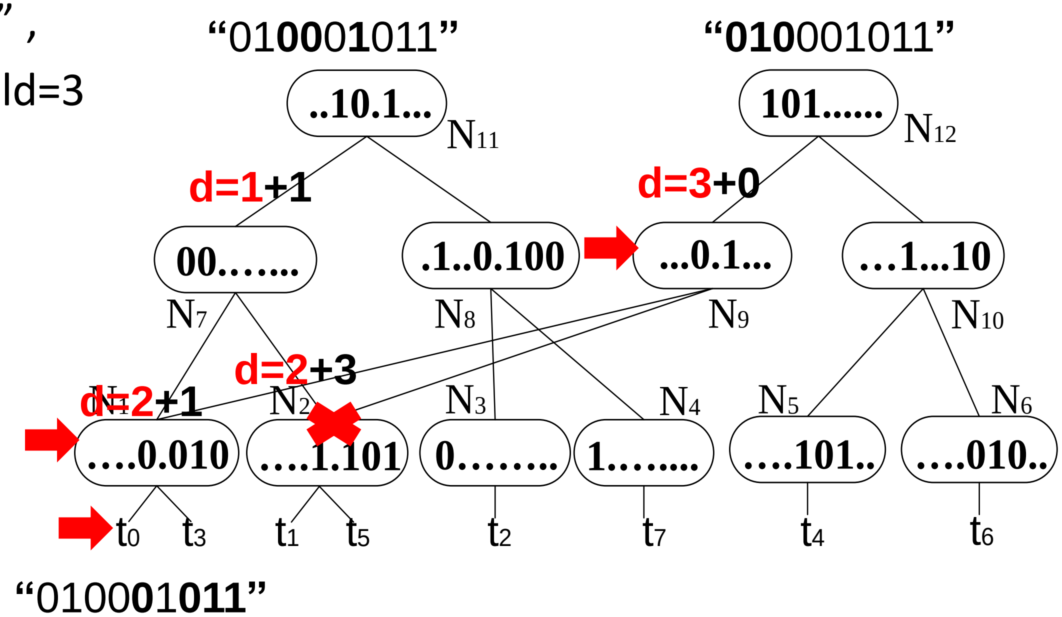
Queue	Qualified tuples ret
N_{11}, N_{12}	$\emptyset$
$N_{12}, [N_7, N_{11}]$	$\emptyset$
$[N_7, N_{11}], [N_9, N_{12}]$	$\emptyset$
$[N_9, N_{12}]$	t_0
$\emptyset$	t_0

HA-Index based

- Hamming-Select
  - Searching from top to bottom (Breadth-First-Search or Depth-First-Search)
  - Given Binary T="010001011" ,

  and Hamming-distance threshold=3
  - Running Example

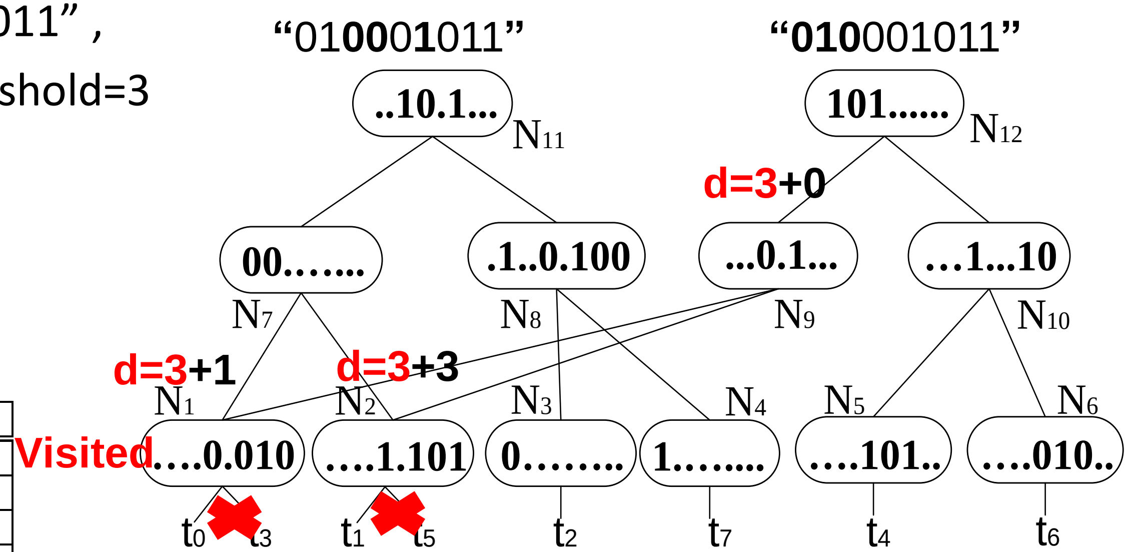
Queue	Qualified tuples ret
N_{11}, N_{12}	$\emptyset$
$N_{12}, [N_7, N_{11}]$	$\emptyset$
$[N_7, N_{11}], [N_9, N_{12}]$	$\emptyset$
$[N_9, N_{12}]$	t_0
$\emptyset$	t_0



HA-Index based

- Hamming-Select
  - Searching from top to bottom (Breadth-First-Search or Depth-First-Search)
  - Given Binary T="010001011" ,

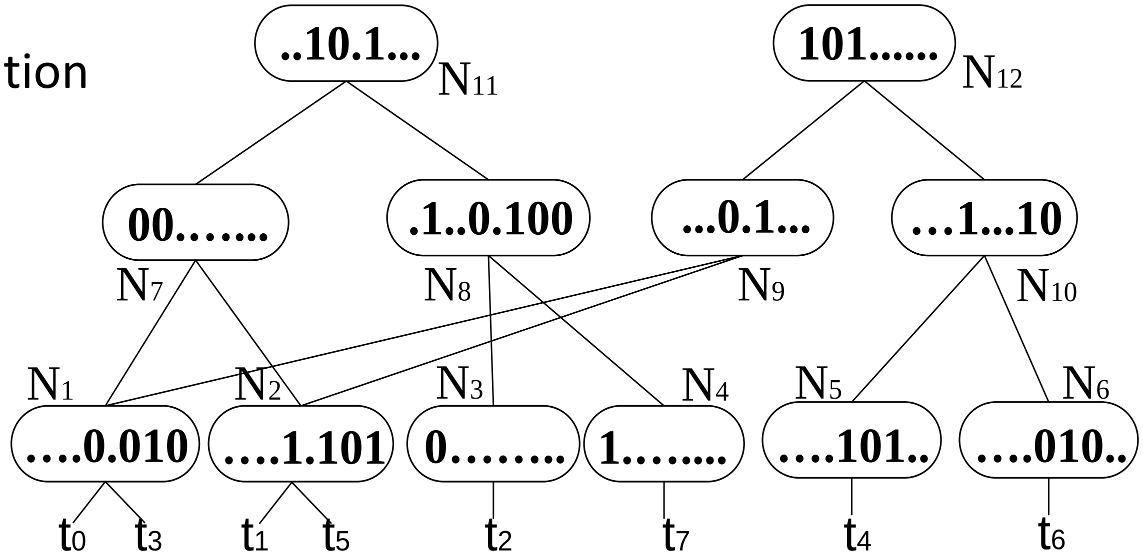
  and Hamming-distance threshold=3
  - Running Example



Queue	Qualified tuples ret
N_{11}, N_{12}	$\emptyset$
$N_{12}, [N_7, N_{11}]$	$\emptyset$
$[N_7, N_{11}], [N_9, N_{12}]$	$\emptyset$
$[N_9, N_{12}]$	t_0
$\emptyset$	t_0

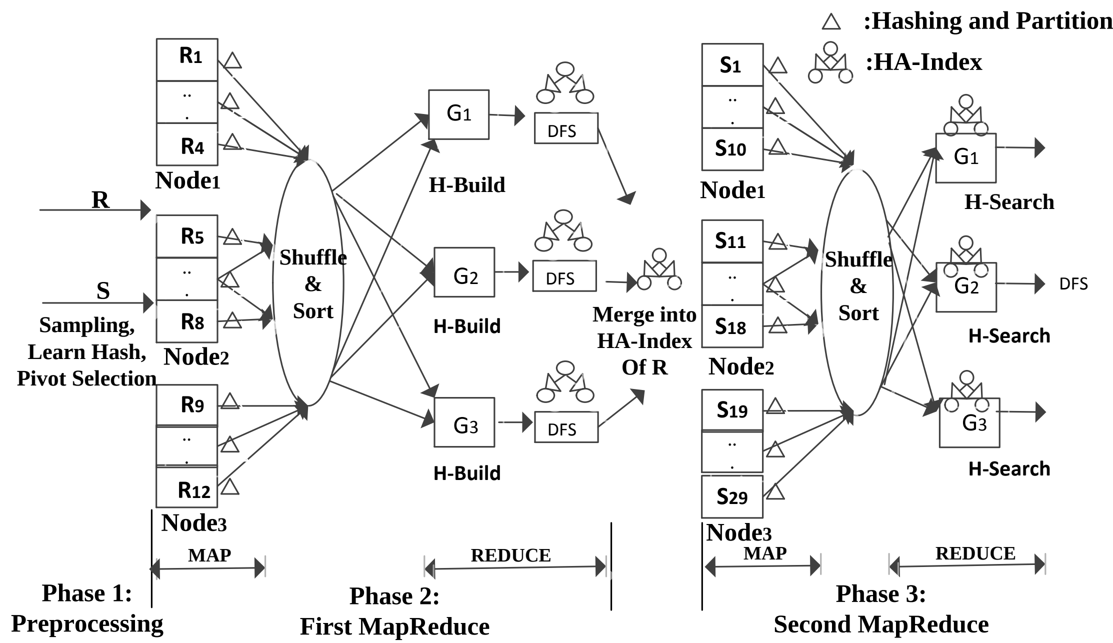
HA-Index based

- Delete Operation
  - Depth-First Search from top to bottom
- Insert Operation
  - Similar to delete operation



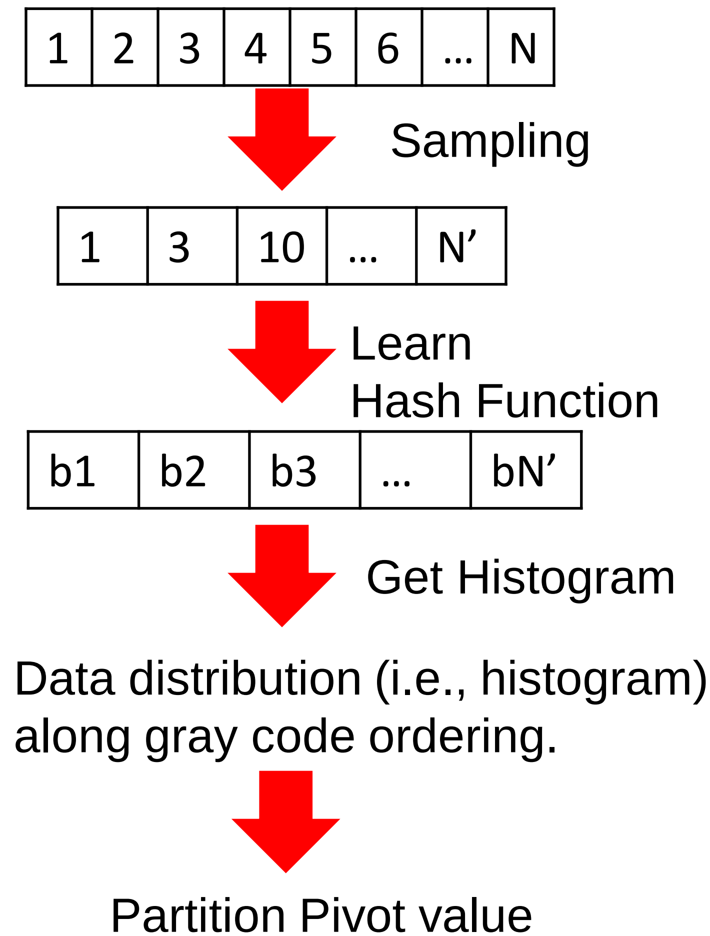
Hamming-join over MapReduce

- Framework
  - Phase 1: Sampling data to learn data partitioning rule
  - Phase 2: Parallel construction of the HA-Index
  - Phase 3: Parallel Hamming-join



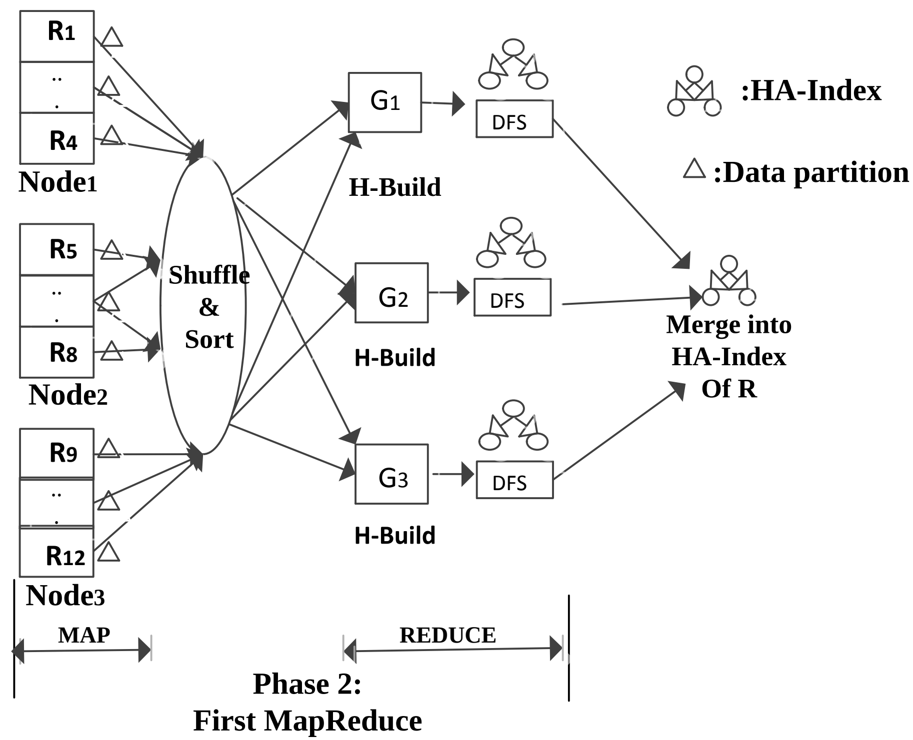
Hamming-join over MapReduce

- Phase 1: Preprocessing
  I. Reservoir sampling from R and S
  II. Learn the Hash Function to map high dimensional data into binary code, i.e., spectral hashing function
  III. Map sample data into binary code, sort the data via gray code ordering
  IV. Get the partition pivot from the sorted binary code
- **Guarantees**
  – Each partition has equal amount of data
  – Data of R and S are sorted via gray code ordering



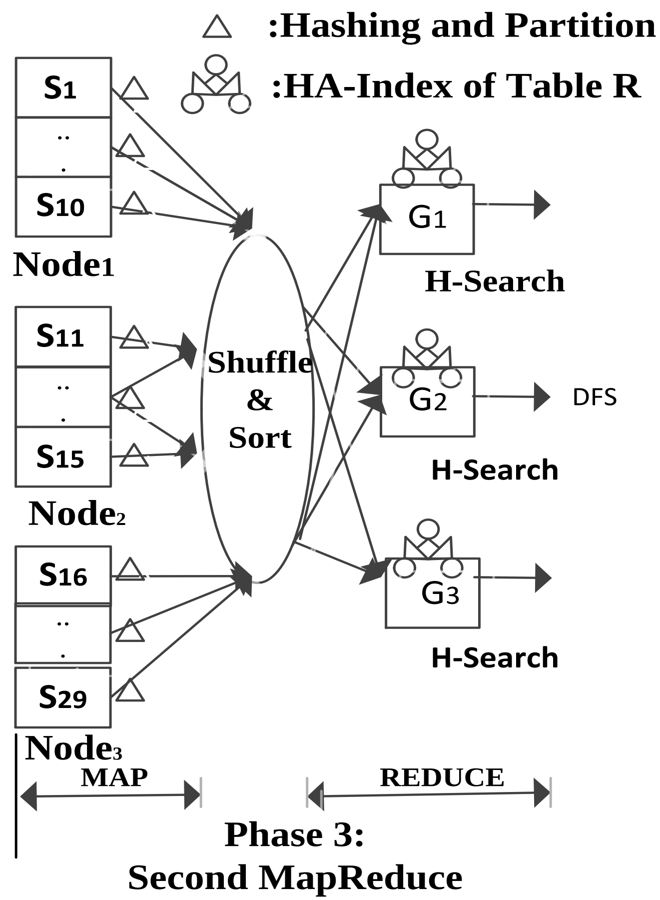
Hamming-join over MapReduce

- Phase 2: HA-Index Building
  - Partition data via the partitioning rule from sampling data
  - Parallel building of HA-Index in each reducer
  - Post-processing step to merge local HA-indexes into one global HA-Index. For example:
    - Merge internal nodes with the same binary codes and relink the pointer
    - Merge leaf nodes with the same binary codes



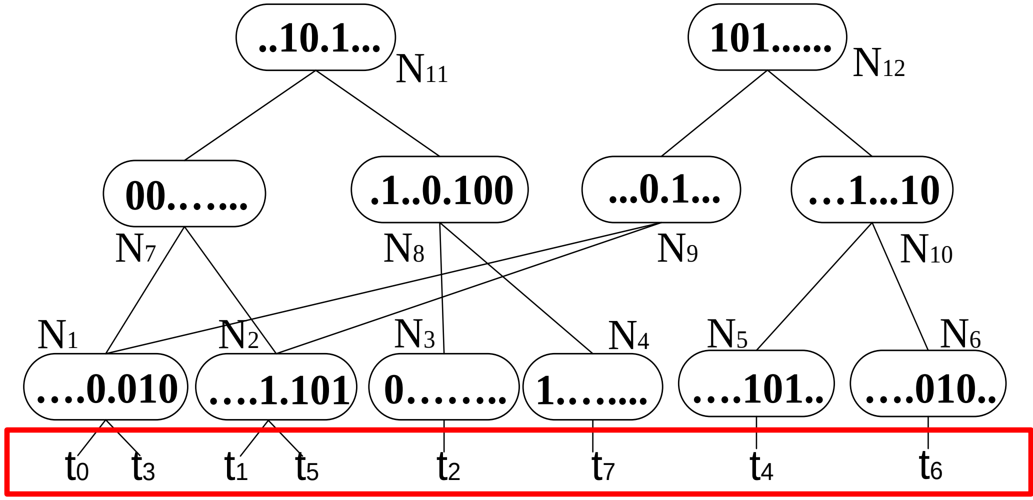
Hamming-join over MapReduce

- Phase 3: Hamming-join
  - Option A:
    - Suppose the number of tuples in Table R is not big enough, and it is affordable to broadcast HA-Index into each server
  - Broadcast HA-Index of Table R into each server, and local Hamming-Join with data of Table S in each server



Hamming-join over MapReduce

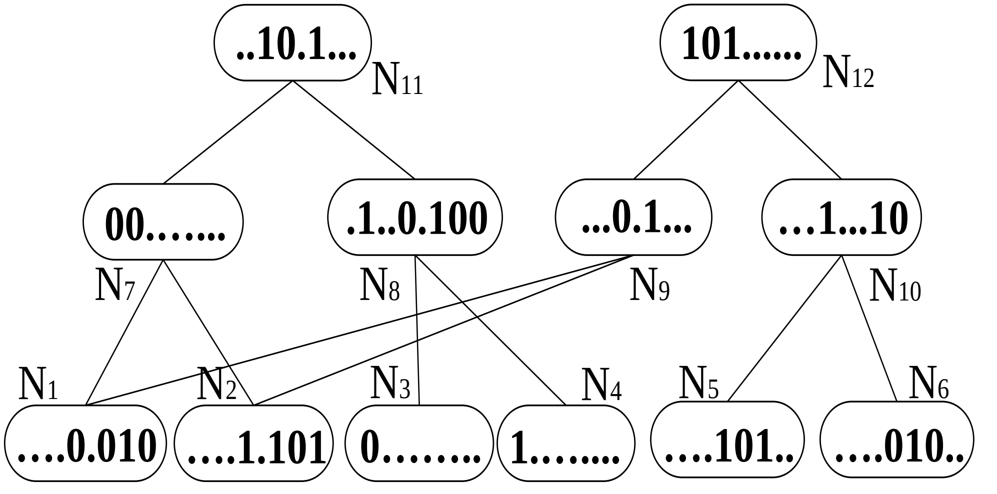
- Phase 3: Hamming-join
  - Option B:
    - Leaf nodes of HA-Index dominate the storage space of HA-Index
  - HA-Index Example:
    - Leaf node: 251 MB
    - Non-leaf: 64 MB



No Leaf Nodes

Hamming-join over MapReduce

- Phase 3: Hamming-join
  - Option B:
    - Hamming-join gets the qualifying binary codes
    - Post-processing step to find the qualifying ID
  - Example:
    - Hamming-join S0 and HA-Index (No-leaf nodes)= S0, “00110011”, “11000001”, “01100011”
    - Post-processing step to find “00110011” = R3, “11000001”=R5, “01100011”=R10
    - Post-processing step can be Hash-join or MapReduce based inner join



Experimental Evaluation

- Effect of parameters
  - Hamming select threshold
  - HA-Index window size and depth
- Scalability of proposed approach
  - For big data more than 20 million
  - Converted to binary data from high dimensional data (images, text)
- Speedup vs. state-of-art Hamming query approaches
  - Google's Mutli-hash Table (WWW 07), Hengine (ICDE 2011)
- Speedup w.r.t. exact and approximate kNN
  - Searching high dimensional data

Experimental Evaluation: Datasets

- **NUS-WIDE:**
  - 269,648 Web images
  - Use 225-D block-wise color moments as the image features
- **Flickr**:
  - Crawled 1 million images
  - Extracted 512 features via the GIST Descriptor
- **DBPedia**:
  - Extracted 1 million documents
  - Applied standard NLP techniques
  - We use LDA model to extract topics and keep 250 topics per document

Experiments

- Hamming distance range query baselines:
  - **NestLoop** - Naive approach to linearly XOR and count the binary data
  - **MultiHashTable**:
    - State-of-the-art to search binary codes for similarity hashing
    - Uses multiple-hash tables to reduce the linear search cost
    - Limit to 4 (MH-4) and 10 (MH-10) hash tables to avoid memory overflow.
  - **Hengine:**
    - Most recent work
    - Improve the MultiHashTable approach in query time and memory usage
  - **Radix-Tree** based approach
  - **Static HA-Index** (SHA-Index)
  - **Dynamic HA-Index** (DHA-Index)}

Note: SHA-Index(32) or DHA-Index(32) = Length of the binary code is 32 bits

Experiment

- Effect of HA-Index:
  - Query time

	(a) NUS-WIDE			(b) Flickr			(c) DBPedia		
method	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)
Nested-Loops	16.42	15.22	/	42.97	41.19	/	59.16	53.53	/
MH-4	6.22	0.21	475	16.09	0.60	712	40.28	0.45	819
MH-10	4.91	0.25	531	14.03	0.83	1204	34.46	0.64	1364
HEngine^s	3.53	0.45	210	14.75	1.14	820	36.91	1.91	763
Radix Tree	1.61	0.19	39	3.98	0.64	365	17.64	0.44	352
SHA-Index	0.87	**0.16**	29	1.75	**0.52**	254	3.54	**0.43**	239
DHA-Index	**0.68**	0.18	**28/11**	**0.74**	0.58	**251/63**	**1.07**	0.51	**225/47**

Experiment

- Effect of HA-Index
  - Index update time (delete and insert)

	(a) NUS-WIDE			(b) Flickr			(c) DBPedia		
method	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)
Nested-Loops	16.42	15.22	/	42.97	41.19	/	59.16	53.53	/
MH-4	6.22	0.21	475	16.09	0.60	712	40.28	0.45	819
MH-10	4.91	0.25	531	14.03	0.83	1204	34.46	0.64	1364
HEngine^s	3.53	0.45	210	14.75	1.14	820	36.91	1.91	763
Radix Tree	1.61	0.19	39	3.98	0.64	365	17.64	0.44	352
SHA-Index	0.87	**0.16**	29	1.75	**0.52**	254	3.54	**0.43**	239
DHA-Index	**0.68**	0.18	**28/11**	**0.74**	0.58	**251/63**	**1.07**	0.51	**225/47**

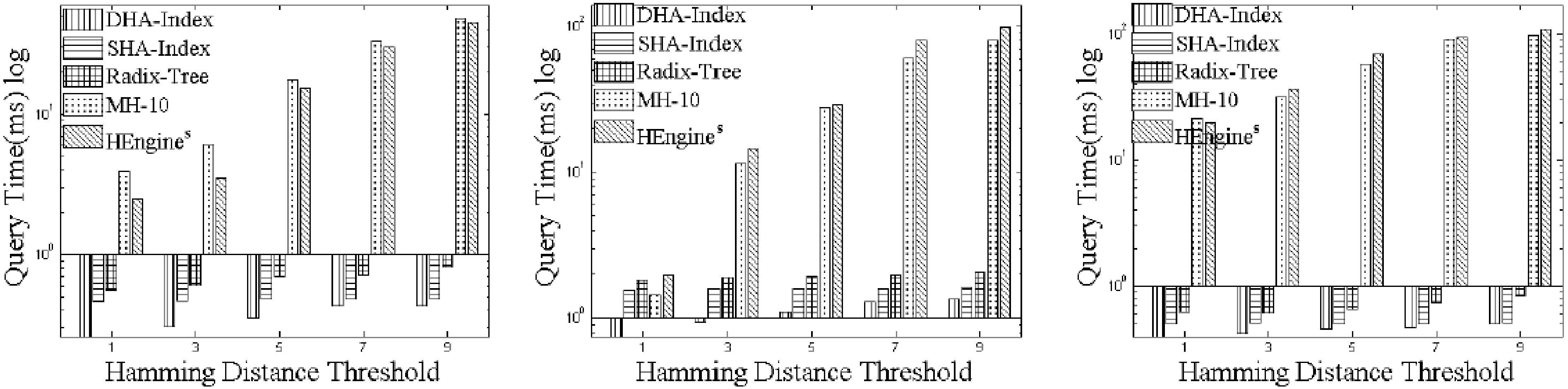
Experiment

- Effect of HA-Index
  - Space usage

	(a) NUS-WIDE			(b) Flickr			(c) DBPedia		
method	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)	query time(ms)	update time(ms)	space usage(mb)
Nested-Loops	16.42	15.22	/	42.97	41.19	/	59.16	53.53	/
MH-4	6.22	0.21	475	16.09	0.60	712	40.28	0.45	819
MH-10	4.91	0.25	531	14.03	0.83	1204	34.46	0.64	1364
HEngine^s	3.53	0.45	210	14.75	1.14	820	36.91	1.91	763
Radix Tree	1.61	0.19	39	3.98	0.64	365	17.64	0.44	352
SHA-Index	0.87	**0.16**	29	1.75	**0.52**	254	3.54	**0.43**	239
DHA-Index	**0.68**	0.18	**28/11**	**0.74**	0.58	**251/63**	**1.07**	0.51	**225/47**

Experiment

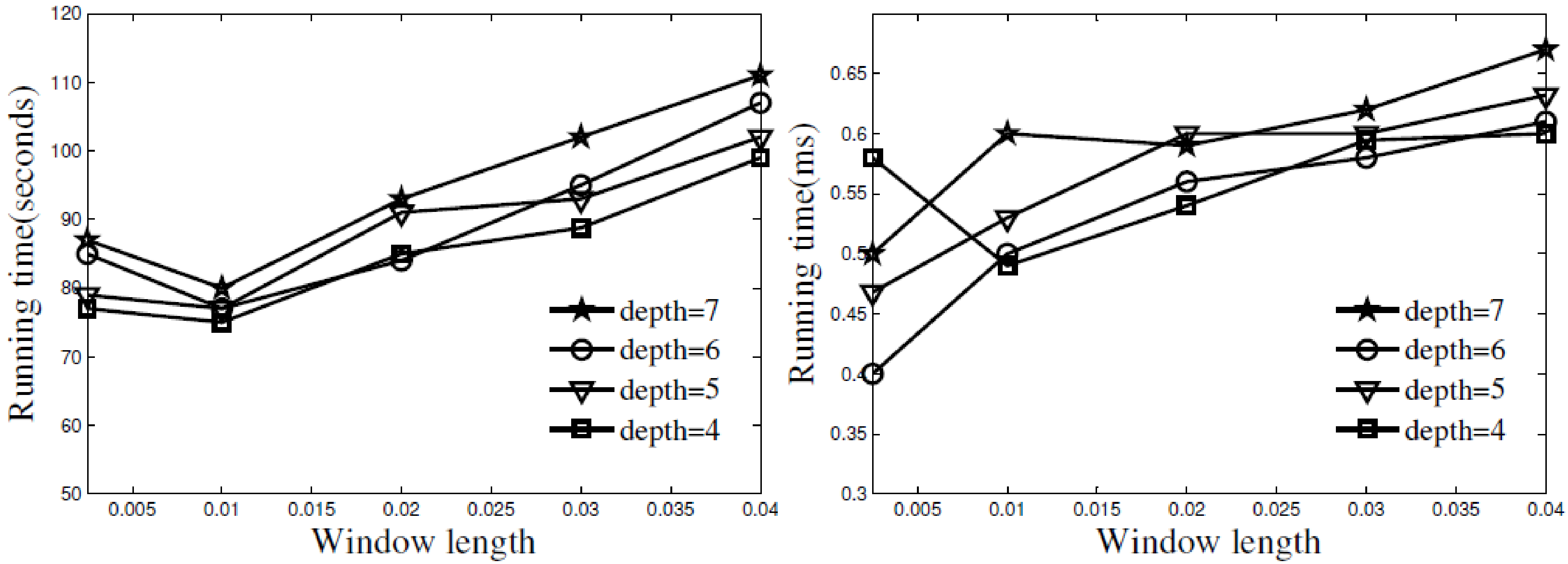
- Hamming-select
  - Effect of Hamming distance range query threshold



(a) NUS-WIDE (b) Flickr (c) DBPedia

Experiment

- Effect of HA-Index parameter
  – Window length and depth



(a) Building Time (b) Query Processing Time.

Experiment

- Hamming distance range query to speedup approximate KNN?
  1. Start from small Hamming distance threshold, i.e., 1, and get the qualifying Hamming distance range query results sets i.e., HSet
  2. KNN search over the HSet
  3. If KNN query sets are smaller than K, enlarge the Hamming distance threshold, and go to Steps 1 and 2

     else exit

  Note: Produce errors but similarity hashing fn guarantees acceptable error bound
- State-of-art approximate KNN approach
  - **Locality-Sensitive Hashing (E2LSH**) is the state-of-art implementation of lSH
  - **LSB-TREE (TODS 2011)** uses Z-order to map high-dimensional data into one-dimensional Z-values, and index the Z-values using a B-tree

Experiment

- Hamming-select: speedup over the kNN

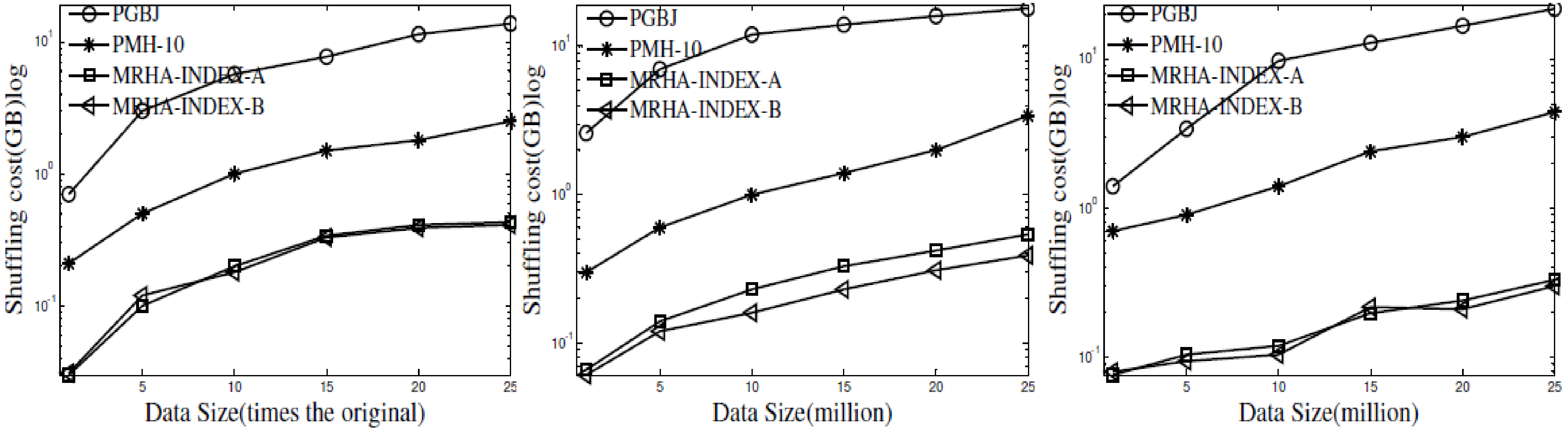
Dataset	Algorithm	Query time(ms)	Index build time
NUS-WIDE	LSH	2400	680(s)
	LSB-Tree(25)	47	37(Hr)
	SHA-Index(32)	2.74	68(s)
	SHA-Index(64)	4.78	97(s)
	DHA-Index(32)	1.64	87(s)
	DHA-Index(64)	2.43	103(s)
Flickr	LSH	340	1080(s)
	LSB-Tree(25)	63	50(Hr)
	SHA-Index(32)	2.21	176(s)
	SHA-Index(64)	3.54	189(s)
	DHA-Index(32)	2.17	210(s)
	DHA-Index(64)	2.88	244(s)
DBpedia	LSH	266	340(s)
	LSB-Tree(25)	59	44(Hr)
	SHA-Index(32)	2.94	150(s)
	SHA-Index(64)	4.88	290(s)
	DHA-Index(32)	2.18	230(s)
	DHA-Index(64)	3.85	310(s)

Experiment

- Evaluate the Map-Reduce Hamming-join
  - **Parallel-exact-KNN-join** (short as PGBJ) is the state-of-the-art approach for performing exact kNN-join over multi-dimensional data in MapReduce,
  - **Parallel Hamming-join via MultiHashTable** (PMH, for short) that handles approximate batch queries for web page duplicate identification
  - **Parallel Hamming-join via Dynamic HA-Index** (MRHA-Index, for short) is the approach introduced in Section 5. Specifically, in terms of the Hamming-join phase, if Option A is used, we term it MRHA-Index-A, and if Option B is used, we term it MRHA-Index-B

Experiment

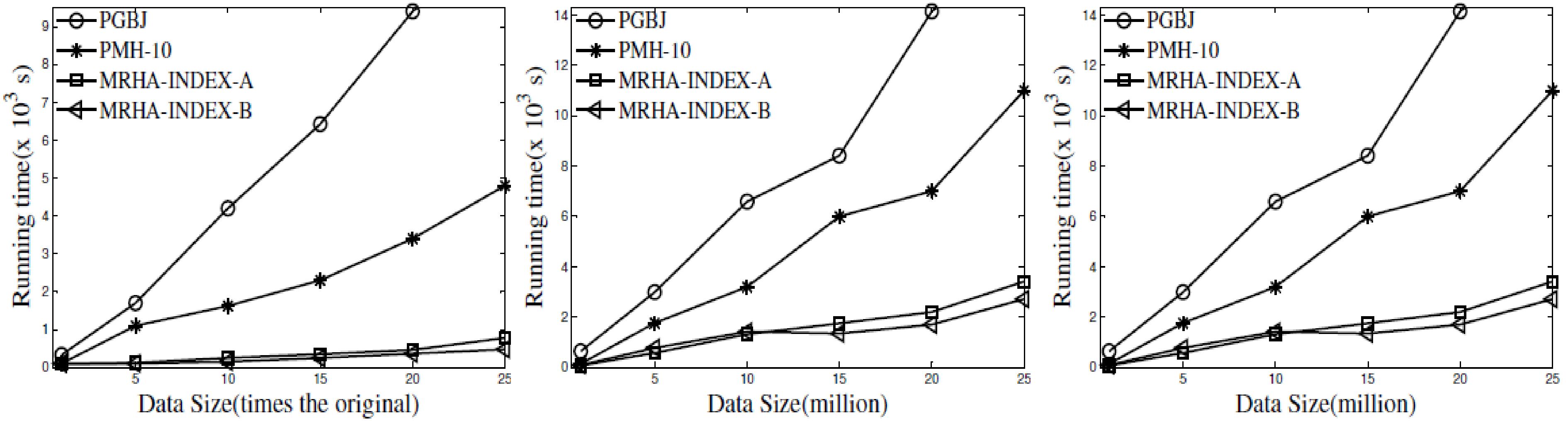
- MapReduce Hamming-join: data shuffle cost



(a) NUS-WIDE (b) Flickr (c) DBPedia

Experiment

- Hamming-join: speedup over the parallel kNN



(a) NUS-WIDE (b) Flickr (c) DBPedia

Experiment

- Summary of experiment results
  - Data Set
    - Image(NUS, Flickr), Text(DBpedia)
  - Effect of HA-Index on Hamming-select
    - Query time: >20x vs Hengine(Liu et al. 2012)
    - Space usage: >30x vs Hengine
  - Effect of HA-Index on Hamming-join over MapReduce
    - Data shuffle cost: >10x vs Parallel-MH (Manku et al. 2007)
    - Speedup: > 10x vs. Parallel-MH

Conclusion

- Proposed several approaches to improve Hamming-select and Hamming-join
- Extensive experimental evaluation using real data to show the performance of newly proposed approaches

Thank you for your attention

Q&A

Reference- LSH

- *Gionis, Aristides and Indyk, Piotr and Motwani, Rajeev: Similarity Search in High Dimensions via Hashing. (Gionis VLDB99)*
- *Alexandr Andoni and Piotr Indyk : Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122. (E2LSH 2008)*
- *Manku, Gurmeet Singh and Jain, Arvind and Das Sarma, Anish: Detecting near-duplicates for web crawling. (ManKu WWW07)*
- *Lee, Hongrae and Ng, Raymond T. and Shim, Kyuseok: Similarity Join Size Estimation Using Locality Sensitive Hashing. (Lee VLDB2011)*
- *Song, Jingkuan and Yang, Yang and Yang, Yi and Huang, Zi and Shen, Heng Tao: Inter-media Hashing for Large-scale Retrieval from Heterogeneous Data Sources. (Song SIGMOD2013)*
- *Yasin N. Silva, Walid G. Aref, Per-Åke Larson, Spencer S. Pearson, and Mohamed H. Ali: Similarity queries: their conceptual evaluation, transformations, and processing. VLDB Journal 2012, pp. 1–26.(Yasin VLDBJ2012)*
- *Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In Proceedings of Computer Vision and Pattern Recognition, 2011.*
- *S. Kumar and R. Udupa. Learning hash functions for cross-view similarity search. In IJCAI, pages 1360{1365, 2011.*

Reference

- J. Buhler. "Provably sensitive indexing strategies for biosequence similarity search." *Journal of Computational Biology* 10(3/4):399-418, 2003. (An earlier version of this work appeared at ACM RECOMB 2002.) (J. Buhler 2003)
- *B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In Proceedings of International Conference on Computer Vision, 2009.*
- *B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. IEEE Trans. Pattern Anal. Mach. Intell., 31(12):2143{2157, 2009.*
- *Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In Proceedings of Neural Information Processing Systems, 2008.*
- *Christian Böhm and Hans-peter Kriegel: A cost model and index architecture for the similarity join. In ICDE 2001*
- *Elke Achtert, Hans-Peter Kriegel, Arthur Zimek: ELKI: A Software System for Evaluation of Subspace Clustering Algorithms. 20th International Conference on Scientific and Statistical Database Management (SSDBM 2008), Hong Kong, China, 2008.*
- *Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu (1996-). "A density-based algorithm for discovering clusters in large spatial databases with noise". Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96).. pp. 226–231*
- R. Sibson : SLINK: Single Link Clustering: Hierarchical clustering algorithm based on single-link connectivity. The Computer Journal 16 (1973)
- M. Zaharia 2013. An Architecture for Fast and General Data Processing on Large Clusters (PhD Disseration).
- Kanungo, Tapas ; Almaden Res. Center, San Jose, CA, USA ; Mount, D.M. ; Netanyahu, N.S. ; Piatko, C.D. An efficient k-means clustering algorithm: analysis and implementation. Pattern Analysis and Machine Intelligence, IEEE Transactions on , Jul 2002