Towards Optimal Utilization of Main Memory for Moving Object Indexing

Bin Cui, Dan Lin and Kian-Lee Tan

School of Computing & Singapore-MIT Alliance,
National University of Singapore
{cuibin, lindan, tankl}@comp.nus.edu.sg

Presented by Donatas Saulys
2007-11-12

Outline

- Motivation
- The IMPACT framework
  - Twin-index
  - Object classification
  - Object migration
  - Memory partitioning
- Experimental results
- Conclusions
- Related work
- Evaluation

Motivation

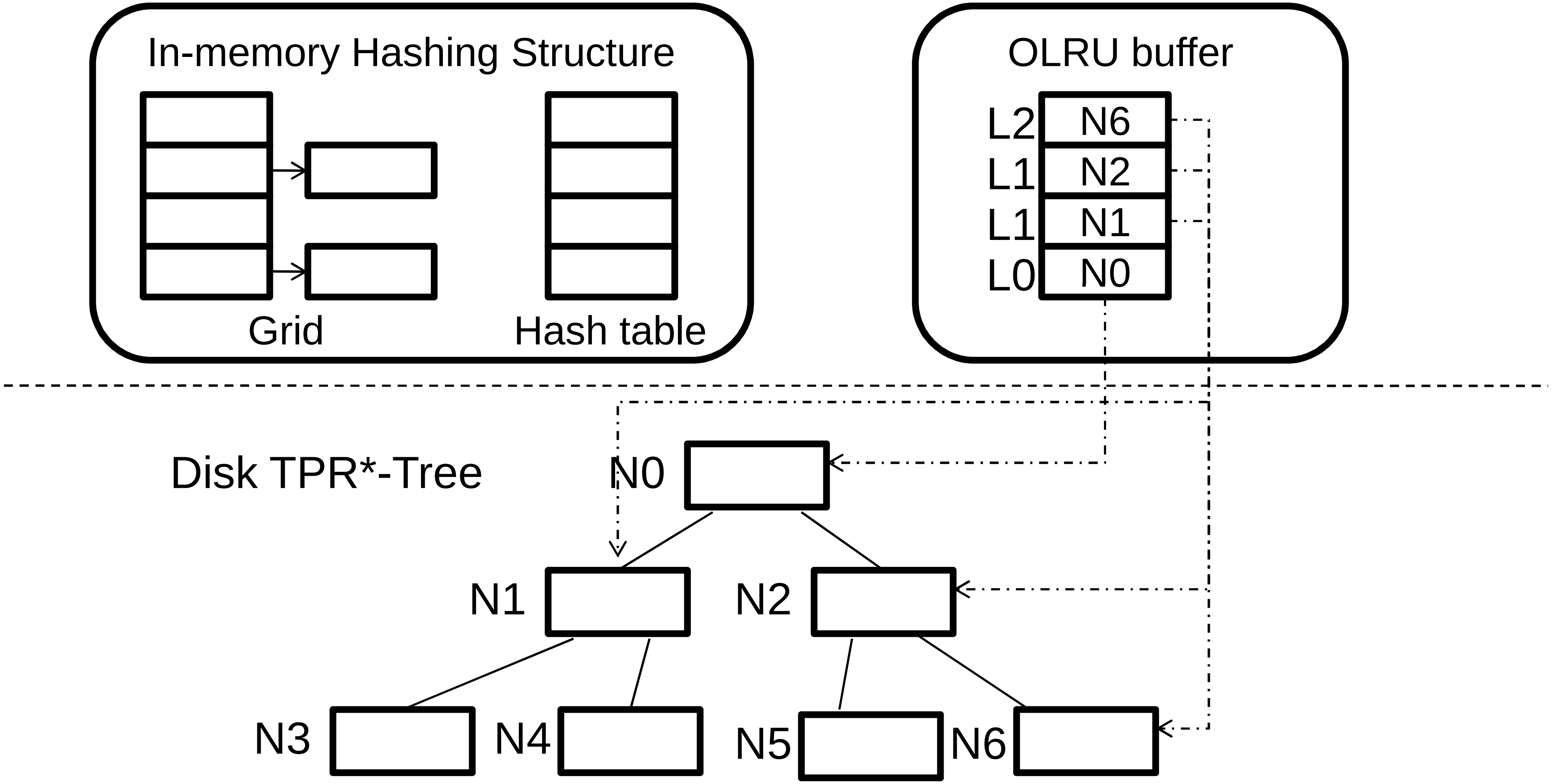
- Tracking moving objects requires a lot of updates
  - Main Memory is much faster than disk
  - Buffering is not enough

- Three observations
  1. object classification
     - active and inactive objects
     - active objects – more updates
  2. most of the objects are inactive
  3. High speed (active) objects degenerate the TPR-Tree index performance

The IMPACT framework

- IMPACT – Integrated Memory Partitioning and Activity conscious Twin-Index
- Twin index
  - A memory resident grid structure for *active* objects
  - A disk based structure for inactive objects (TPR*-Tree)
- Object classification
- Object migration
- Memory partitioning

IMPACT: Structure



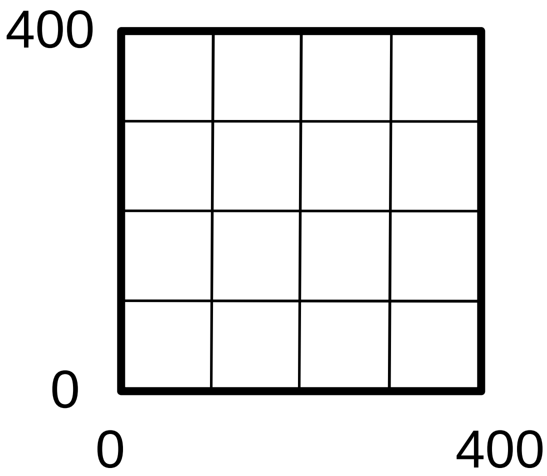
IMPACT: Twin Index

- A memory resident grid structure
  - Stores active objects

Hash table

ID	x	y	t	$\vec{v}$

Grid



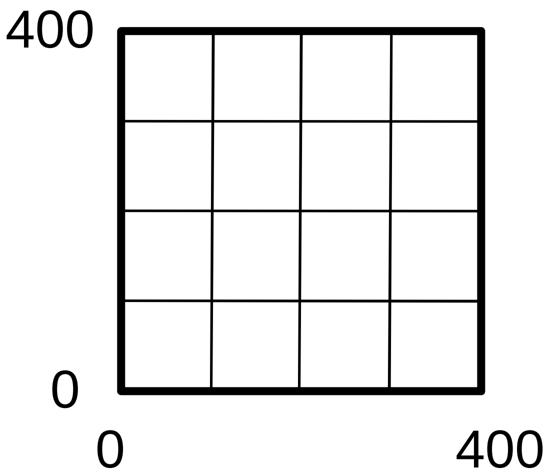
IMPACT: Twin Index

- A memory resident grid structure (example)
  - grid 400x400 (1 grid cell = 100x100),
  - 4 active objects
  - Future query time horizon H = 2

Hash table

ID	**x**	**y**	**t**	$\vec{v}$

Grid



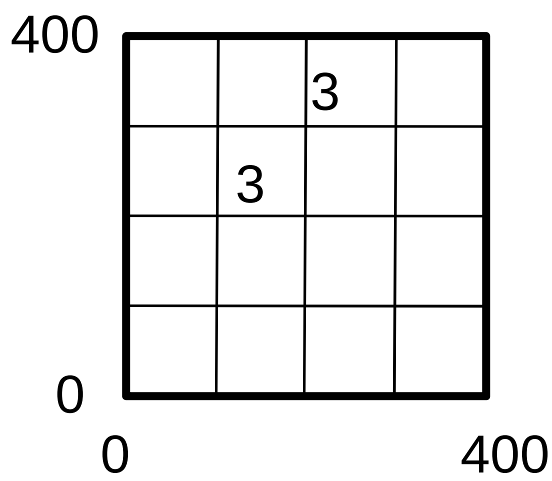
IMPACT: Twin Index

- A memory resident grid structure (example)
  - Object 3 is inserted into the hash table
  - Object 3 is inserted into the grid
  - Future positions of object 3 are inserted into the grid (H=2)

Hash table

ID	x	y	t	$\vec{v}$
3	205	305	0	(-40,-40)

Grid



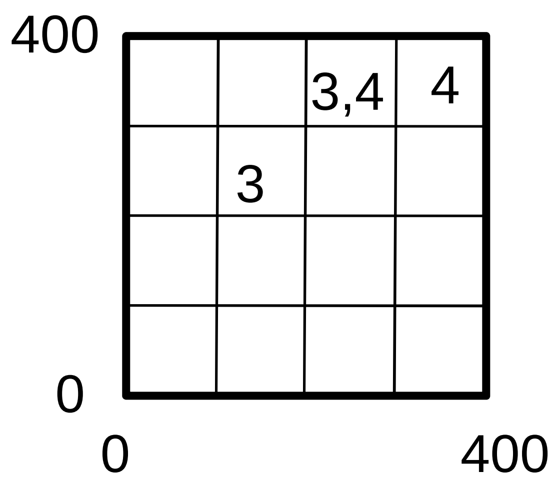
IMPACT: Twin Index

- A memory resident grid structure (example)
  - Object 4 is inserted into the structure

Hash table

ID	**x**	**y**	**t**	$\vec{v}$
3	205	305	0	(-40,-40)
4	260	310	0	(50, 0)

Grid



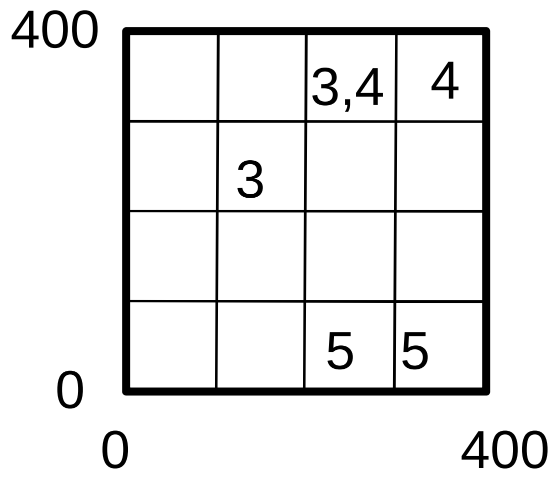
IMPACT: Twin Index

- A memory resident grid structure (example)
  - Object 5 is inserted into the structure

Hash table

ID	x	y	t	$\vec{v}$
3	205	305	0	(-40,-40)
4	260	310	0	(50, 0)
5	330	50	0	(-40, 0)

Grid



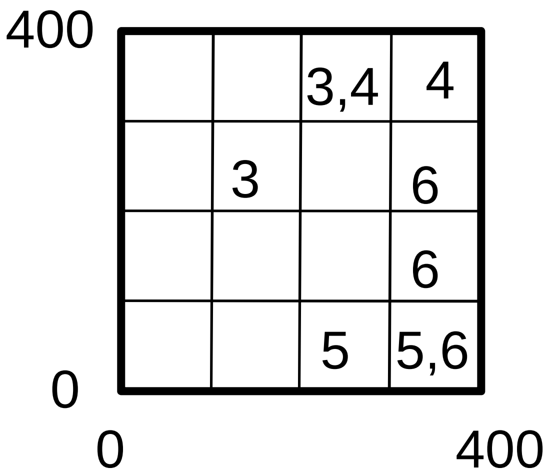
IMPACT: Twin Index

- A memory resident grid structure (example)
  - Object 6 is inserted into the structure

Hash table

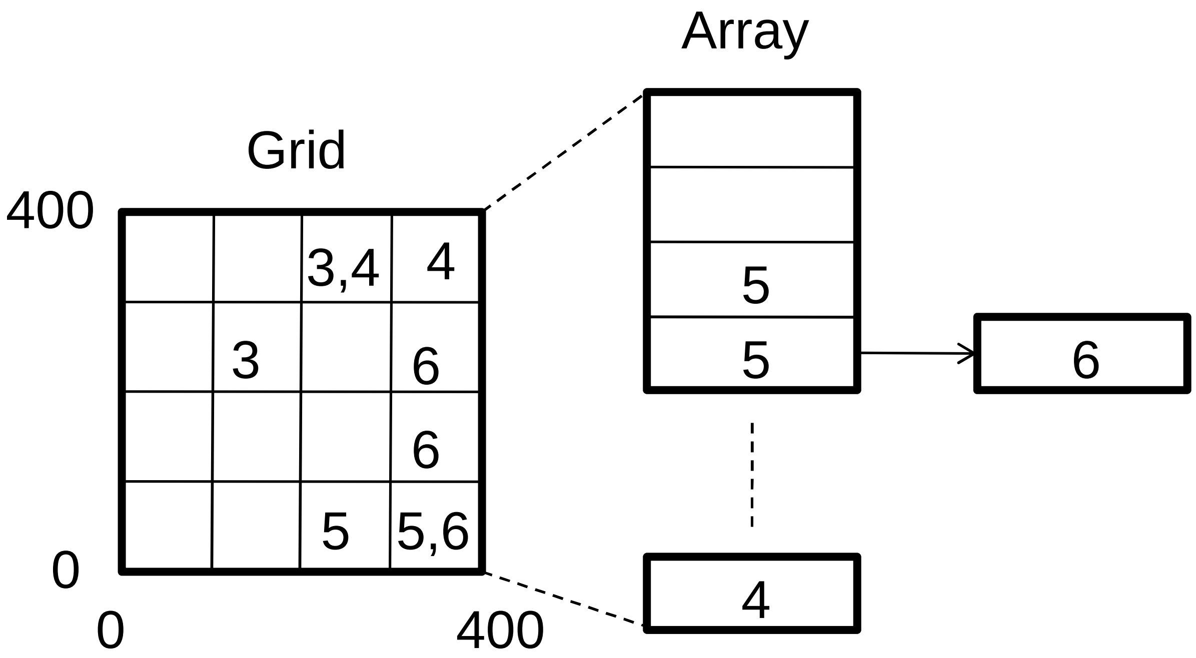
ID	**x**	**y**	**t**	$\vec{v}$
3	205	305	0	(-40,-40)
4	260	310	0	(50, 0)
5	330	50	0	(-40, 0)
6	350	90	0	(0,60)

Grid



IMPACT: Twin Index

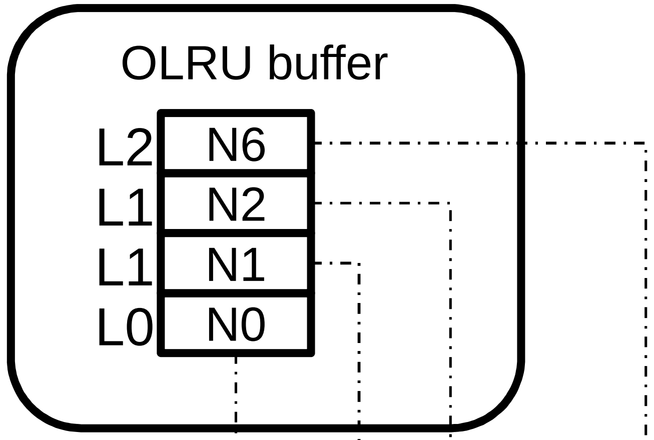
- A memory resident grid structure (example)
  - The grid is stored as an array
  - Objects in the same cell are stored in a bucket (e.g. linked list)

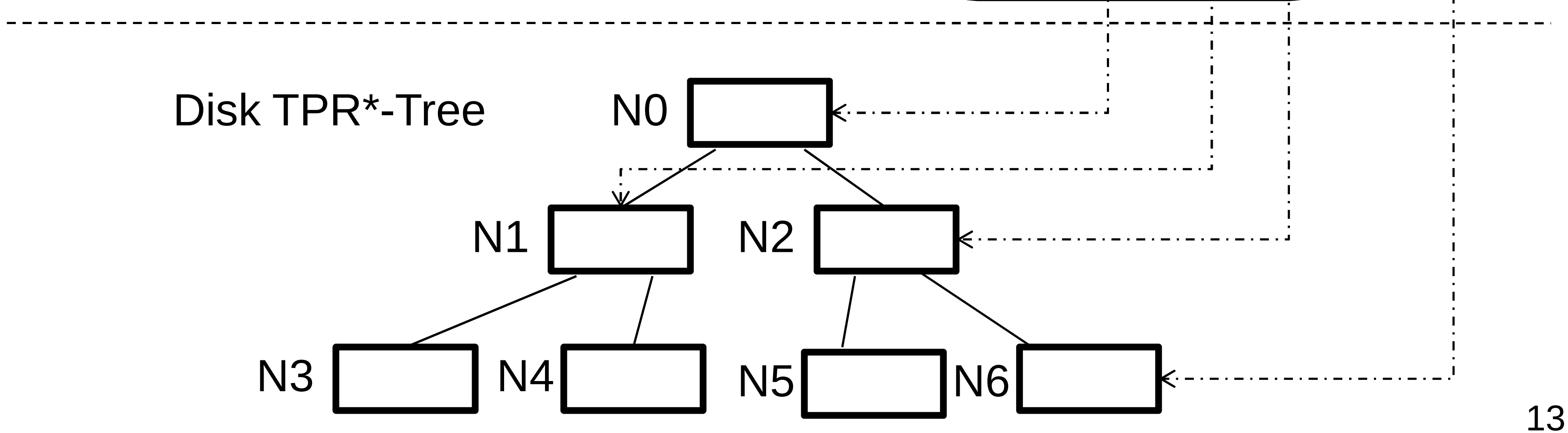


IMPACT: Twin Index

- A disk-based structure (TPR*-Tree)
  - Stores inactive objects

OLRU buffer - *In general, the OLRU scheme allocates the available buffer* according to reference frequency of nodes.





IMPACT: Object classification

- Velocity threshold V
  - Fast objects (active) – huge expansions of MBRs
  - Slow objects (inactive) – no significant influence on TPR*-tree's performance

- Determining V
  - Velocity histogram
  - Determine V according to the histogram and available memory
  - Update the histogram on every update
  - Adjust V periodically (e.g. rush hour)

IMPACT: Object migration

- An active object becomes inactive
  - v(OID) < V
  - e.g. v(3) < V

Grid

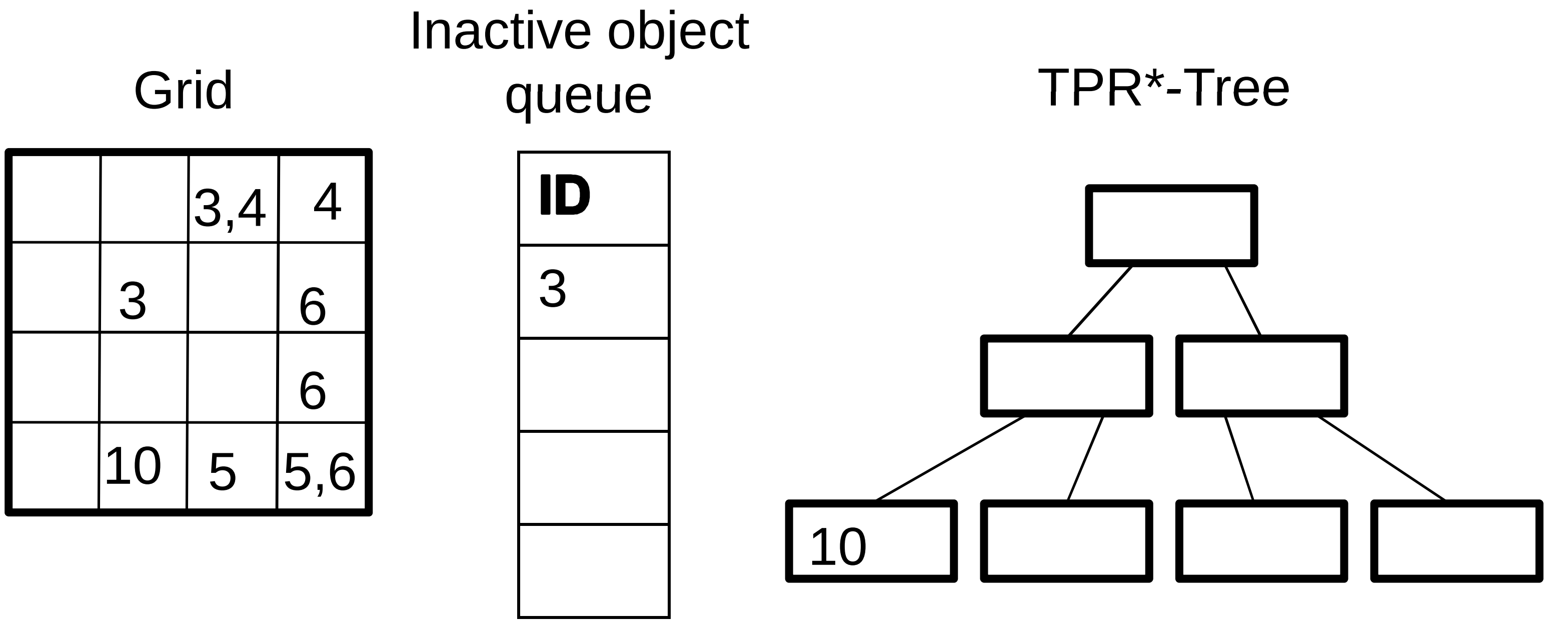
		3,4	4
	3		6
			6
		5	5,6

Inactive object queue

ID
3

IMPACT: Object migration

- An inactive object becomes active
  - v(OID) > V, e.g. v(10) > V
  - There is free memory



IMPACT: Object migration

- An inactive object becomes active
  - v(OID) > V, e.g. v(10) > V
  - There is NO free memory

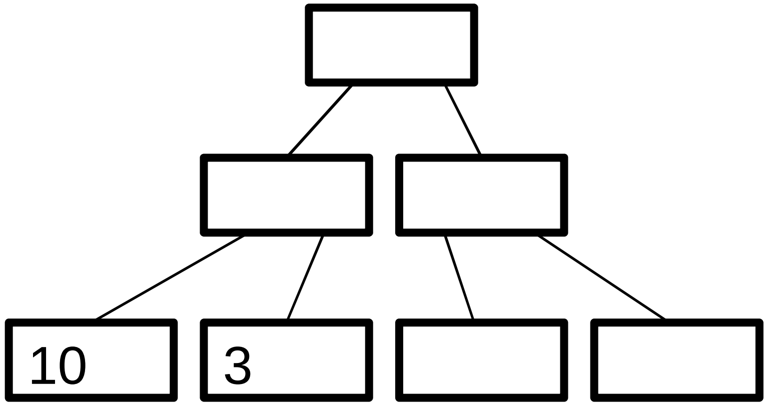
Grid

		3,4	4
	3		6
			6
	10	5	5,6

Inactive object queue

ID
3

TPR*-Tree



- Frequent migration is avoided

IMPACT: Memory partitioning

- Memory allocation
  - For the grid structure
  - For the OLRU buffer of the tree
- What allocation is optimal?
- Cost analysis on Uniformly Distributed Data
  - Buffer the 2 top levels of the TPR*-Tree
  - Allocate the rest to the grid

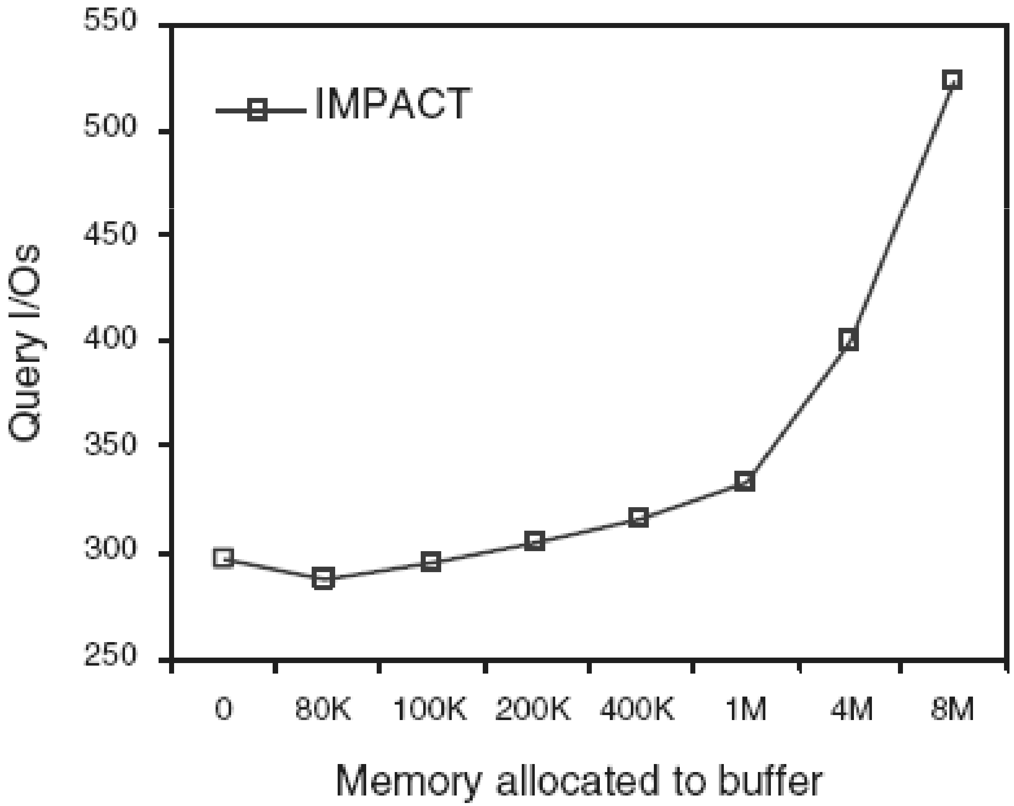
Experimental results

- Experimental settings
  - A default total Main Memory of 8 MB
  - Comparison with TPR*-Tree
    - All main memory used for OLRU buffering
  - 200 range queries (4% of the space)

- Uniform and skew datasets
  - 1000000 points (objects)

Experimental results

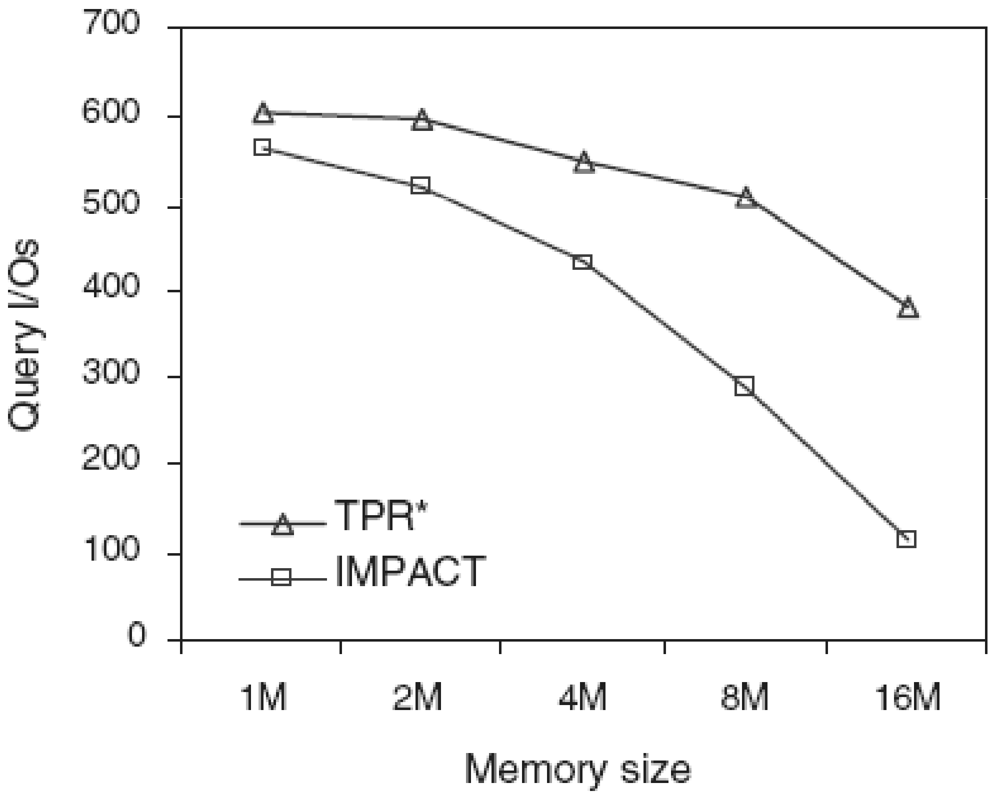
- Effect of Memory Allocation



- More memory for the grid yields better performance
- If all memory for the buffer, then IMPACT~TPR*-Tree
- Optimal – 80K for the buffer (top 2 levels of the tree)

Experimental results

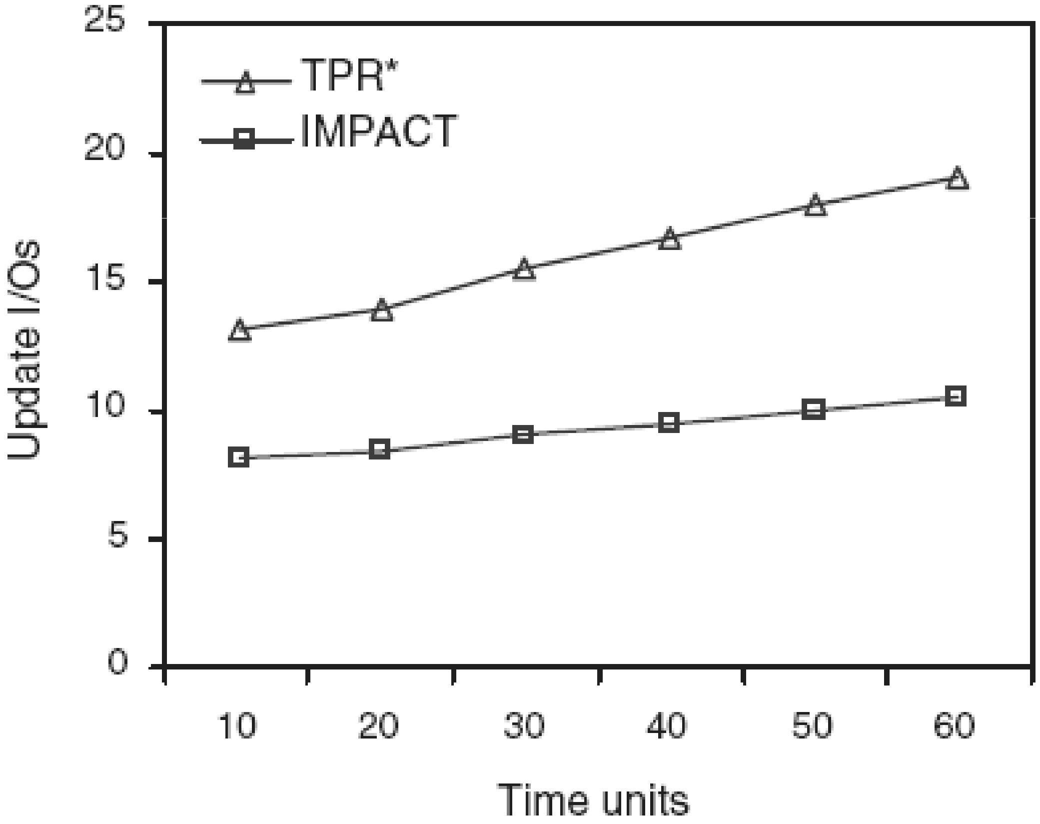
- Effect of Memory Size



- If Total ~ 1M, then IMPACT ~ TPR*-Tree
- If Total > 8M, then IMPACT can be 100% better than TPR*-Tree
- Traditional buffering does not effectively utilize main memory

Experimental results

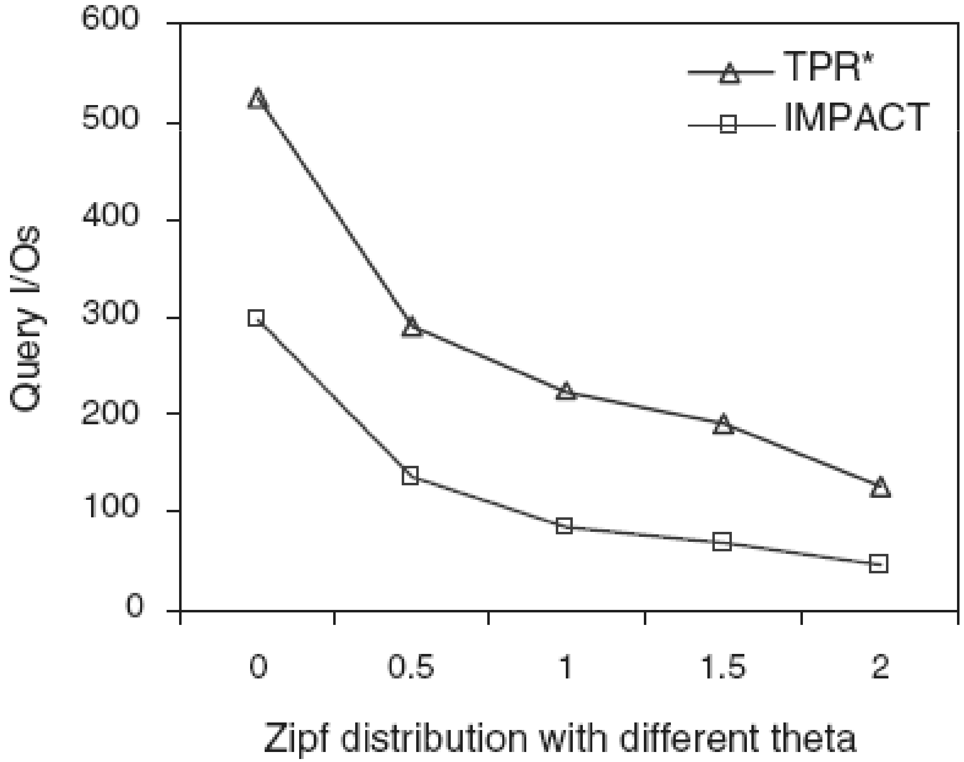
- Effect of The Number of Updates



- Average update cost is increasing over the number of processed updates (time)
- IMPACT efficiency degenerates slower
  - Fast memory updates
  - Less overlap
  - Slower MBR enlargement

Experimental results

- Effect of Varying Velocity Distribution
  - Theta↑ => more inactive objects



- Both indices lead better performance with theta↑
- The active objects are the main bottleneck in both indices
- Handling them in main memory pays off.

Conclusions

- IMPACT framework
  - Motivation - Object classification
  - Twin-index
  - Efficient memory partitioning
    - In-memory grid
    - OLRU buffer for the disk based index
- Experiments show that IMPACT leads to better performance than the TPR*-Tree

Related work

- DAT4 project –
  Indexing Moving Objects in Main Memory
  - ONLY Main Memory is used (no disk)
  - Predictive queries are also supported (handled differently)
  - Hash table for fast access
  - Grid structures, no Trees

- DAT5 project
  - Using a Tree structure instead of a Grid

Evalution

- Good points
  - Well written, easy to read
  - Memory partitioning strategy based on analysis
  - Nice experimental result graphs and explanations

Evalution

- Could be improved
  - Not enough details
    - on grid maintenance when time evolves
    - on predictive queries in the grid
    - how the query performance is affected by the low number of updates (TPR-Tree)
  - Too few algorithms
    - Range query?
    - Velocity threshold V adjustment?

Grid, t = 01?

		3,4	4
	3		6
			6
		5	5,6

The END