

Query Optimization Techniques for Partitioned Tables

Herodotos Herodotou, Nedyalko Borisov, Shivnath Babu

Presented by Bairong Lei

Feb 28, 2013

Overview

- Background
- Problem
- Proposed Solution
- Experiment
- Critiques and Discussion

Background

- Purpose of table partitioning
 - Separate unneeded data during query processing
 - Data access parallelism for query execution
 - Separation of concern for data maintenance
 - Prioritizing data management basing on data popularity

Background

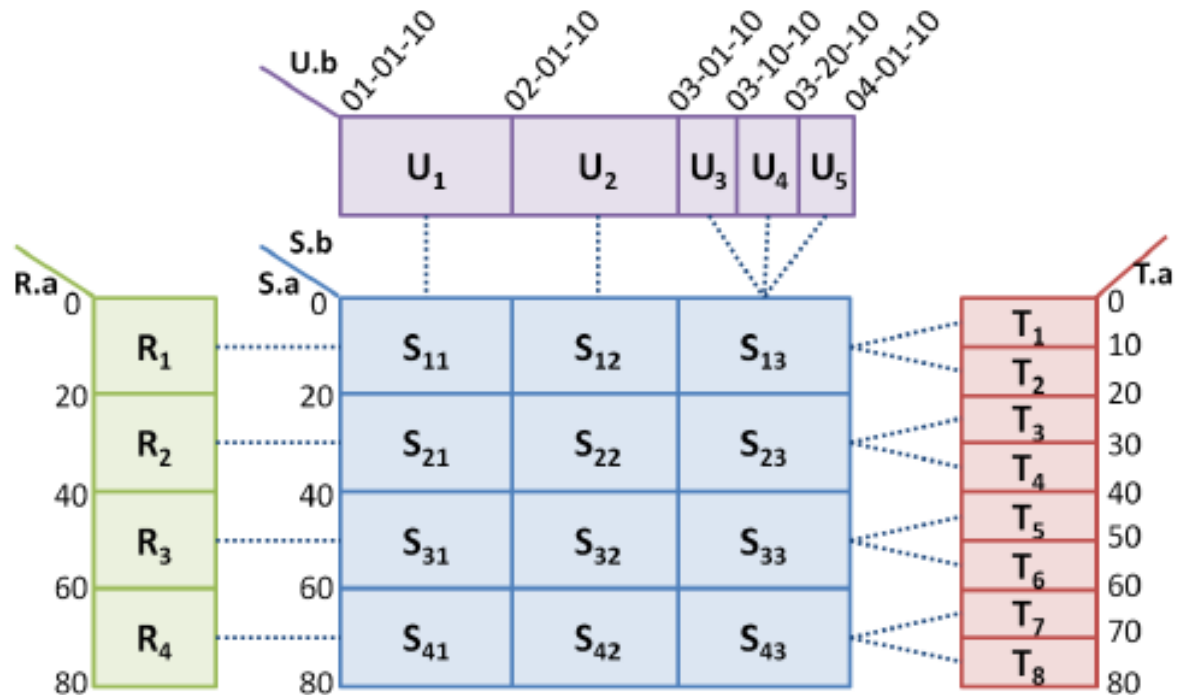
- Table partitioning techniques
 - Horizontal partitioning
 - Hash
 - Range
 - Vertical partitioning
 - Hierarchical (multidimensional) partitioning
- Partitioning and Optimizers
 - Pruning of partitions at plan execution time (IBM DB2)
 - Dynamic partitioning with 1-to-1 join relationship (Oracle)
 - Predicate optimization (move predicates to obtain better plans)

Problem

- The gap between query optimization and user control of table partitioning (need to adapt optimizer for repartitioning due to data popularity)
- Optimizer used to handle only the restricted partitioning schemes defined by the DBA on base tables.
- Consideration of plan space explosion

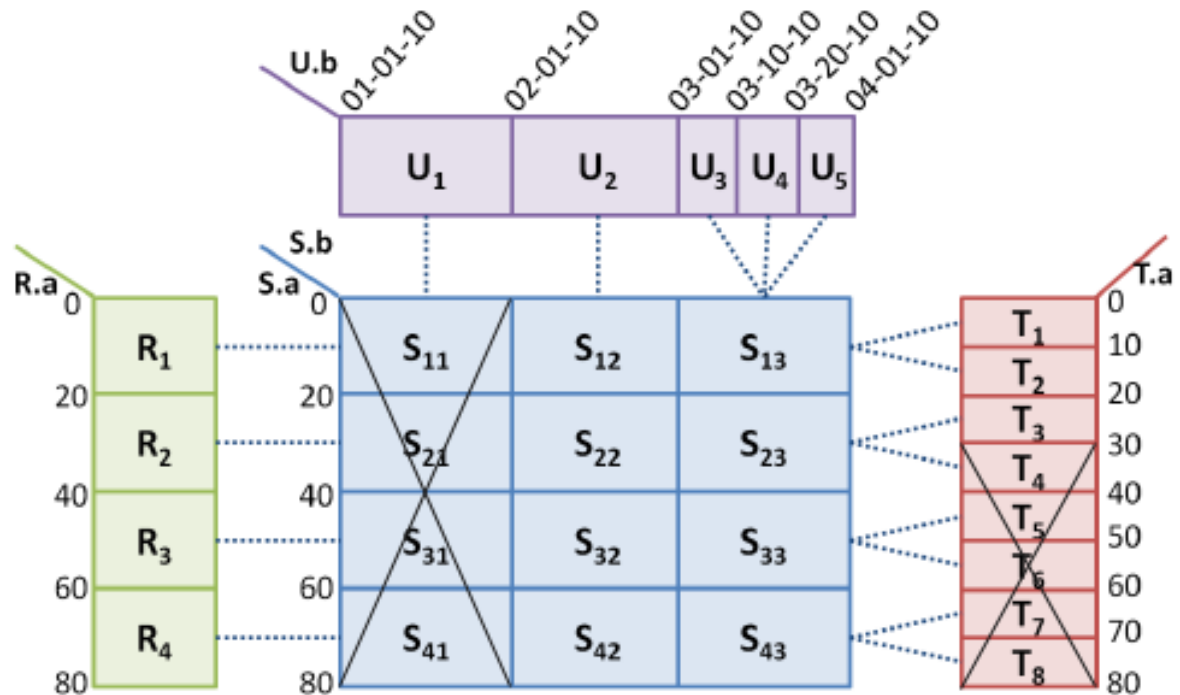
Problem

- Example:
Select *
From R, S, T
Where R.a = S.a
and S.a = T.a
and S.b >= 02-15-10
and T.a < 25;



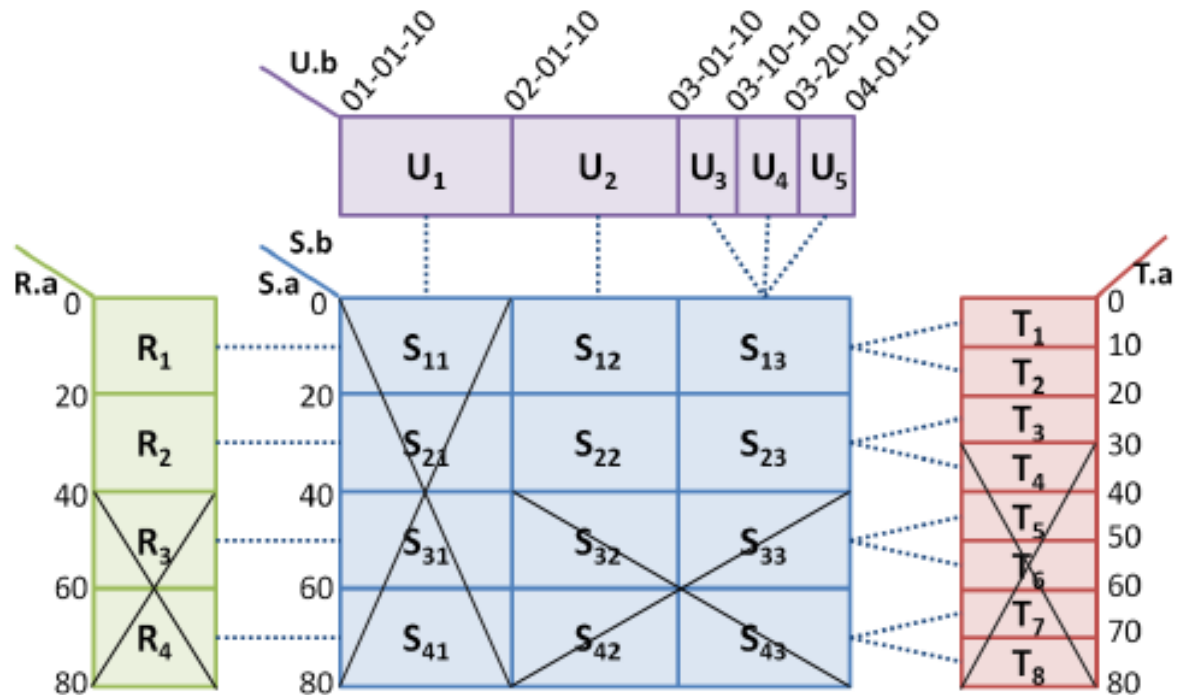
Problem

- Example:
Select *
From R, S, T
Where R.a = S.a
and S.a = T.a
and S.b >= 02-15-10
and T.a < 25;

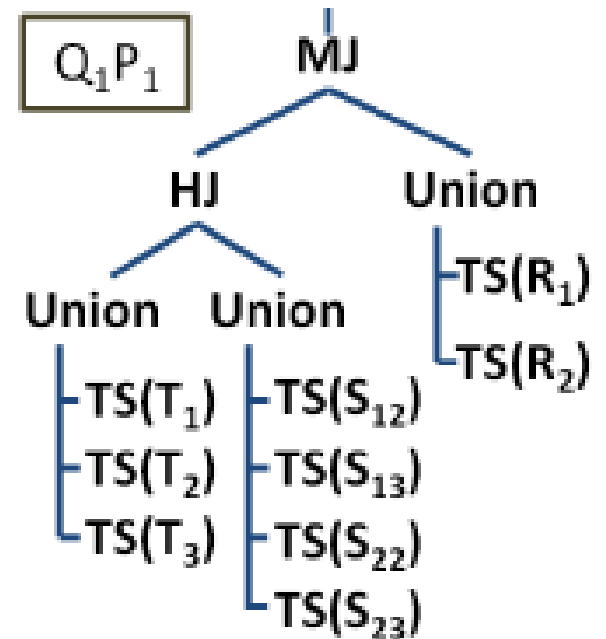
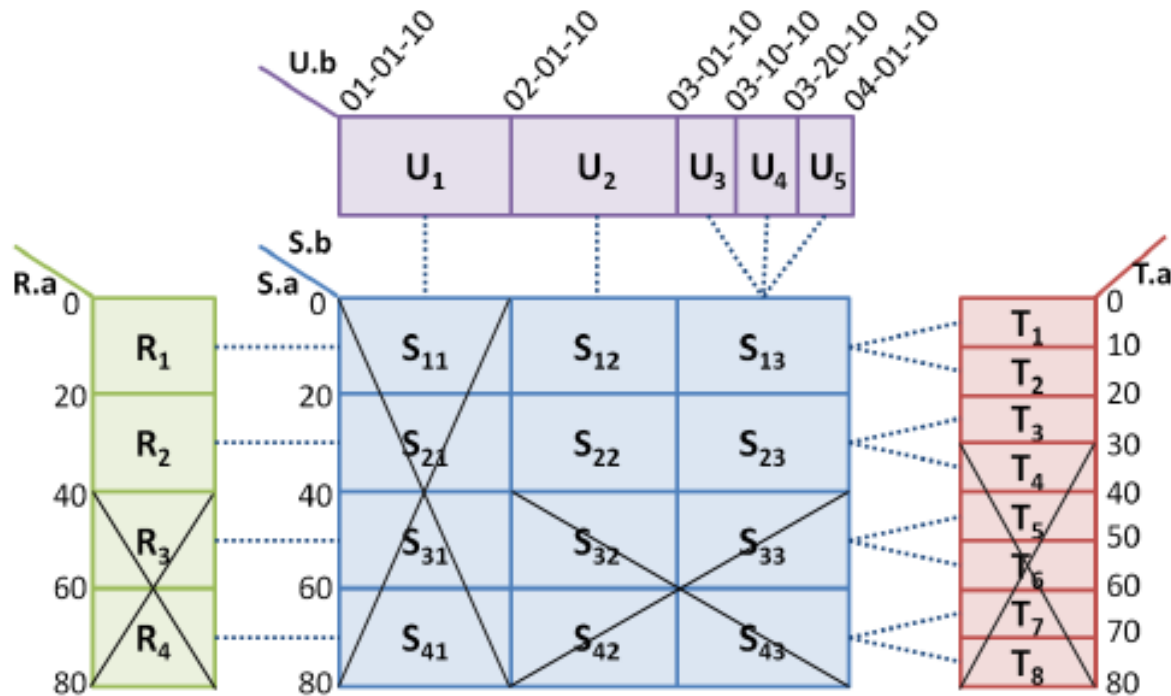


Problem

- Example:
Select *
From R, S, T
Where $R.a = S.a$
and $S.a = T.a$
and $S.b \geq 02-15-10$
and $T.a < 25$;



Problem



Proposed Solution

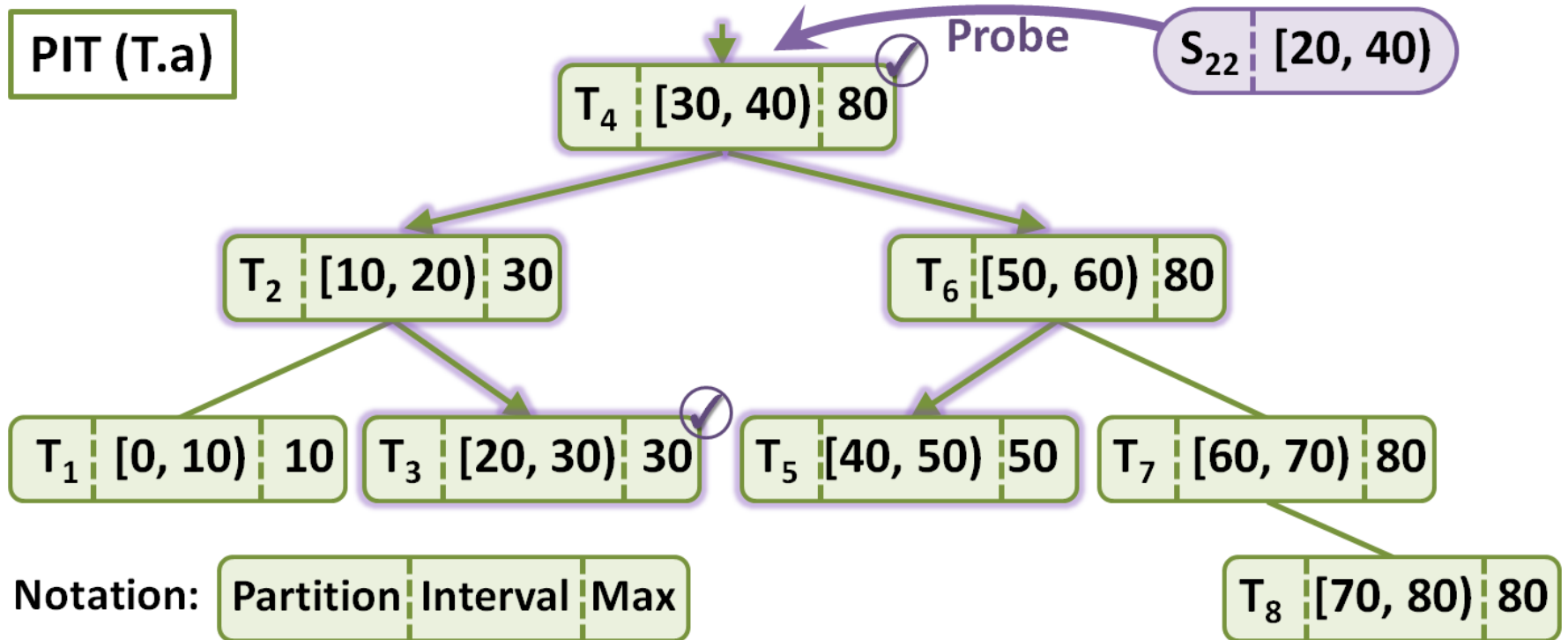
- Bottom-up optimizer
 - Best access join path for each pair of child table
 - Join path selection from the bag unions of the child tables between two parent tables
 - Four phases: applicability testing, matching, clustering and path creation

Proposed Solution

- Applicability testing
 - Join conditions and partitioning condition match checking
- Matching
 - Join selection between child tables based on partition conditions
- Clustering
 - Cluster child tables to reduce join paths creation overhead
- Path creation
 - Generate the path representing the union of the best child-join paths

Matching Phase

- Partition Index Tree



Matching Phase

Algorithm for performing the matching phase

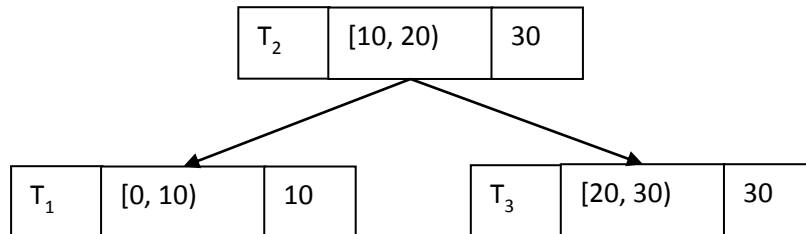
Input: Relation R, Relation S, Join Condition

Output: All partition-wise join pairs (R_i, S_j) that can produce join results

```
For each (binary join expression in Join Condition) {  
    Convert all partitioning conditions to intervals;  
    Build PIT with intervals from partitions of R;  
    Probe the PIT with intervals from partitions of S;  
    Adjust matching result based on logical AND or OR semantics  
    of the  
        Join Condition;  
}
```

Matching Phase

- From previous query, un-pruned partitions:
 - $R_1, R_2; S_{12}, S_{13}, S_{22}, S_{23}; T_1, T_2, T_3$
- Example: $S \bowtie T$
 - $S_{12}, S_{13}, S_{22}, S_{23}; T_1, T_2, T_3$



Output:

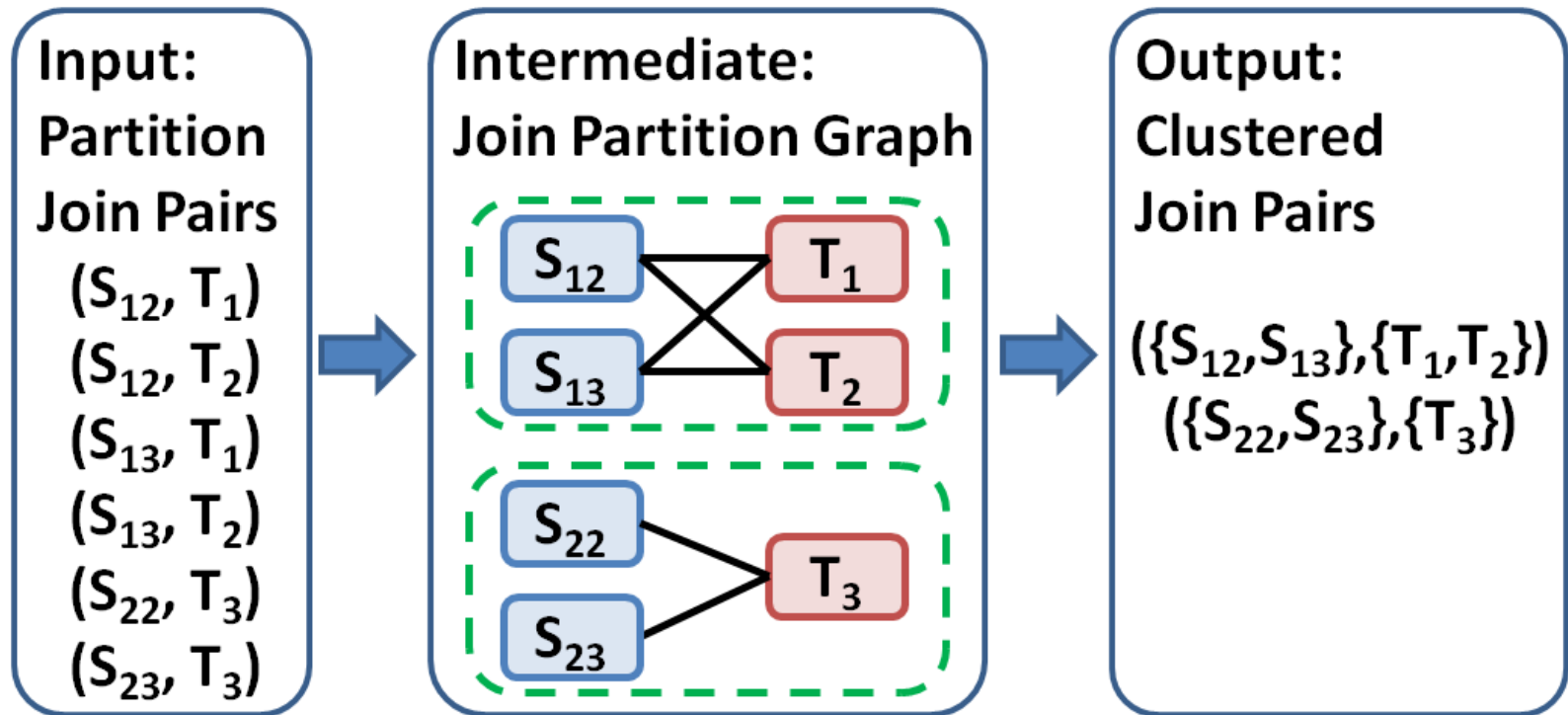
$\{(S_{12}, T_1), (S_{12}, T_2),$
 $(S_{13}, T_1), (S_{13}, T_2),$
 $(S_{22}, T_3), (S_{23}, T_3)\}$

Matching Phase

- Complexity of a PIT:
 - N: number of partitions for building a PIT
 - M: number of partitions for probing a PIT
 - Building a PIT requires $O(N \log N)$ time
 - Probing a PIT requires $O(\min(N, k \times \log N))$, where k is the number of matching intervals
 - Space overhead for PIT is $\Theta(N)$

Clustering Phase

- To reduce the number of join pairs to avoid both optimization and execution inefficiencies



Clustering Phase

- **Clustering metric:**

For an $R \bowtie S$ join, two child tables S_j and S_k of S will be clustered together iff there exists a child table R_i of R such that the matching phase outputs the join pairs (R_i, S_j) and (R_i, S_k)

- **Clustering algorithm:**

Input: Partition join pairs (output of matching phase)

Output: Clustered join pairs (which will be input to path creation phase)

Build a bipartite join graph from the input partition join pairs where:

Child tables are the vertices, and

Partition join pairs are the edges;

Use Breadth-First-Search to identify connected components in the graph;

Output a clustered join pair for each connected component;

Path Creation and Selection Phase

- Three approaches on extending bottom-up optimizer to find the optimal plan
 - Extended enumeration
 - Treating partitions as a physical property
 - Treating partitions as a logical property

Path Creation and Selection Phase

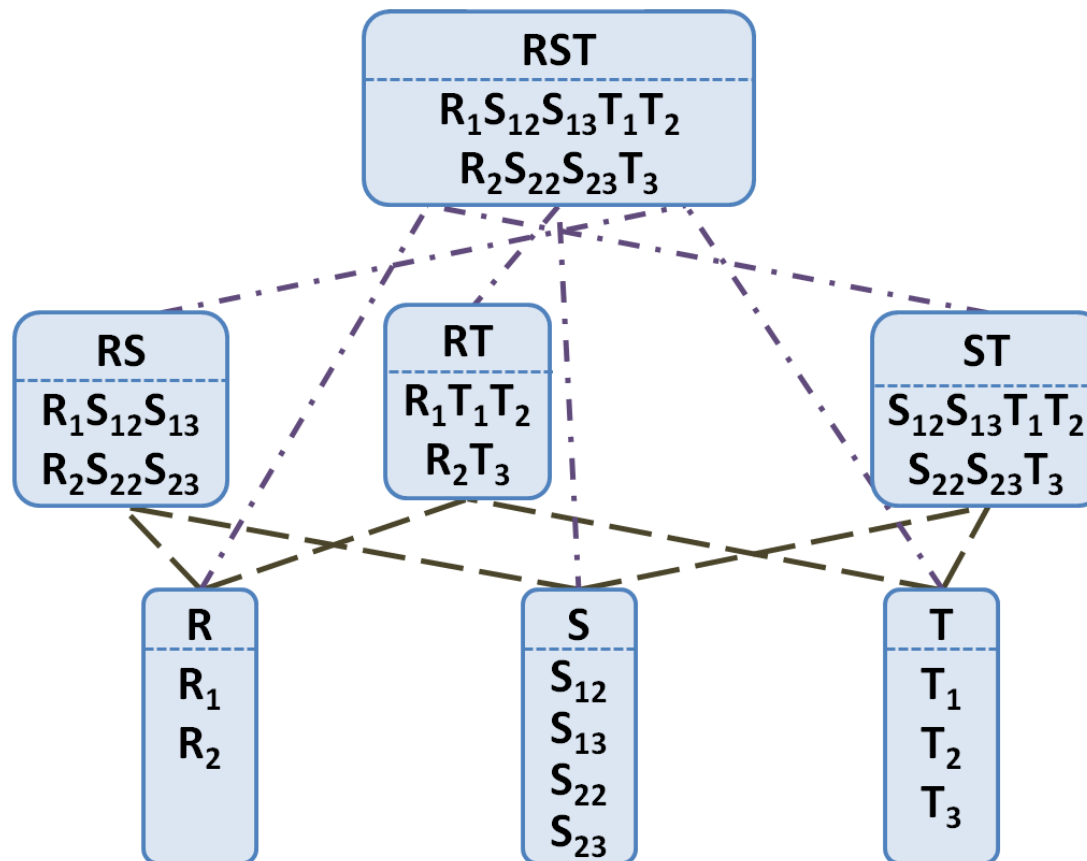
- Extended enumeration
 - Take the join $S \bowtie T$ as an example:
 - Create join paths for $(S_{12} \cup S_{13} \cup S_{22} \cup S_{23}) \bowtie (T_1 \cup T_2 \cup T_3)$
 - Also create join paths for $(S_{12} \cup S_{13}) \bowtie (T_1 \cup T_2)$ and $(S_{22} \cup S_{23}) \bowtie T_3$
 - Find the corresponding best paths and create the union P_u of the best child-join path. Keep P_u if it is the best and consider it when moving on to larger joins
 - This approach will NOT always find the optimal plan.

Path Creation and Selection Phase

- Treating partitions as a physical property
 - Interesting partitions are partitions on attributes reference in equality join and on grouping attributes.
 - Make use of partitioning information from parent tables to create more than two way child joins
 - Optimizer consider each combination of interesting join order and interesting partitions only at parent table level
 - It will NOT always find the optimal plan since it disregards interesting orders independently for each child join

Path Creation and Selection Phase

- Treating partitions as a logical property
 - Each logical relation maintains a list of logical child relations



Path Creation and Selection Phase

- Treating partitions as a logical property (Con't)
 - Disregard the join order or the join operators used in the physical execution
 - All child-join paths with interesting orders are considered during path creation for higher child joins
 - Paths with interesting orders for a single child join can be used later up the lattice, independent from all other child joins of the same parent relation (Property 3)

Path Creation and Selection Phase

- Treating partitions as a logical property (Con't)
 - The optimizer will consider plans where different child joins of the same parent relation can have different join orders and/or operations (Property 4)
 - Optimality guarantee: By treating partitioning as a logical property, our bottom-up optimizer will find the optimal plan in the extended plan space (Property 5)
 - Separate enumeration process of the logical relations from the construction of physical plans

Experiment

- Environment and setup
 - PostgreSQL 8.3.7 optimizer
 - Amazon EC2 nodes of m1.large type
 - 7.5GB RAM, dual-core 2GHz CPU and 850 GB HD
 - 10 TPC-H queries with 2-way up to 8-way joins
- Evaluation metrics
 - Execution times
 - Optimization times
 - Memory usage

Experiment

- Evaluation methods
 - Three categories of optimizers
 - Basic
 - Intermediate
 - Advanced (as described in this paper)
 - DBA-Controlled schemes
 - Constrained schemes
 - Size and number of partitions
 - Stress test on a synthetic benchmark
 - Application of clustering algorithm

Experiment

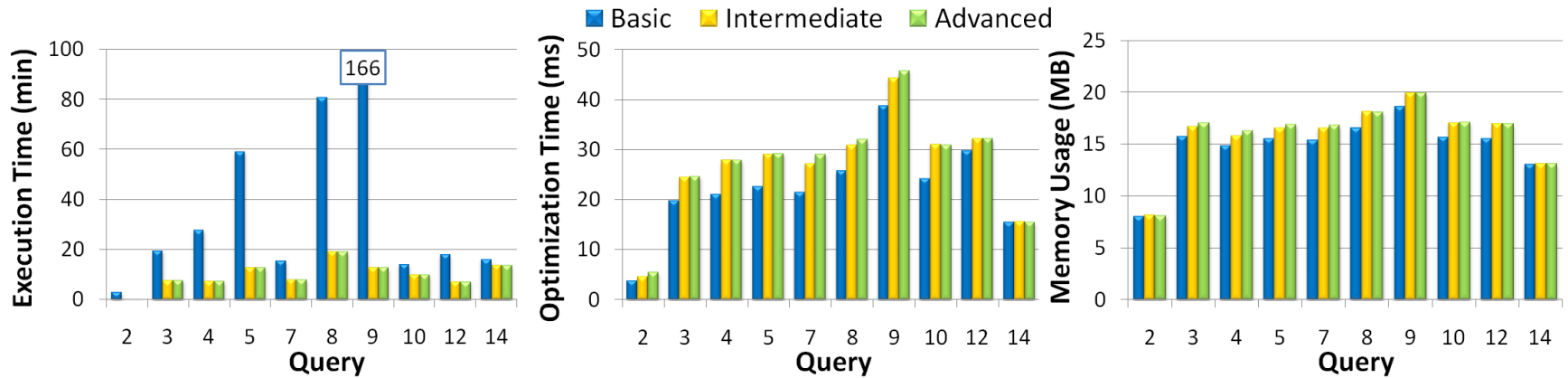
Name	Features
Basic	Per-table partition pruning only (like MySQL and PostgreSQL). Our evaluation uses the PostgreSQL 8.3.7 optimizer as the Basic optimizer
Intermediate	Per-table partition pruning and one-to-one partition-wise joins (like Oracle and SQLServer). The Intermediate optimizer is implemented as a variant of the Advanced optimizer that checks for and creates one-to-one partition-wise join pairs in place of the regular matching and clustering phases
Advanced	Per-table partition pruning and all the join optimizations for partitioned tables as described in the paper

Experiment

- Partitioning schemes

Partition Scheme	Table	Partitioning Attributes	Number of Partitions
PS-P	orders	o_orderdate	28
	lineitem	l_shipdate	85
PS-J	orders	o_orderkey	48
	lineitem	l_orderkey	48
	partsupp	ps_partkey	12
	part	p_partkey	12
PS-B	orders	o_orderkey, o_orderdate	72
	lineitem	l_orderkey, l_shipdate	120
	partsupp	ps_partkey	12
	part	p_partkey	6
PS-C	orders	o_orderkey, o_orderdate	36
	lineitem	l_orderkey, l_shipdate	168
	partsupp	ps_partkey	30
	part	p_partkey	6
	customer	c_custkey	6

Results from DBA-Controlled Schemes

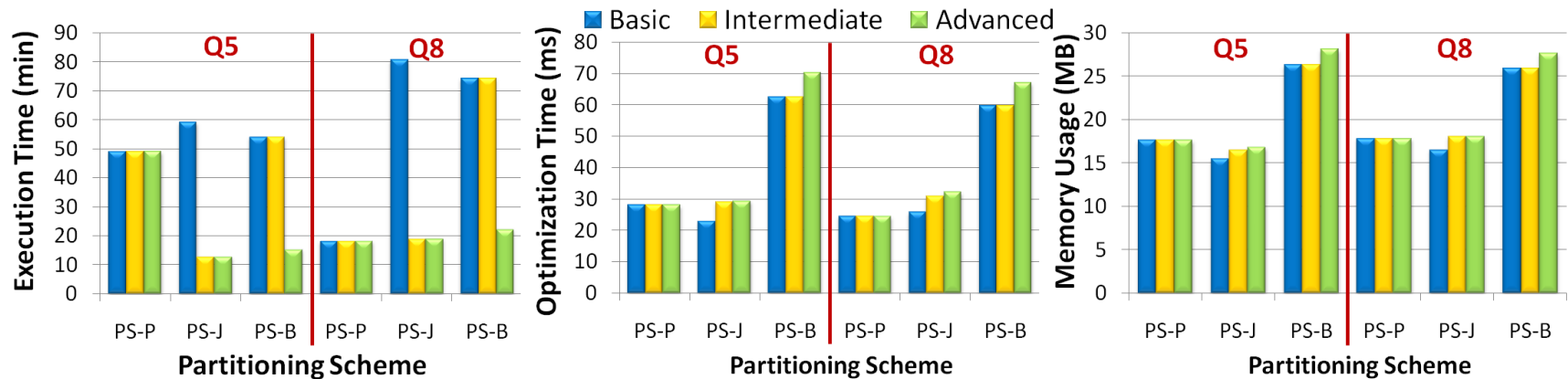


Execution times, optimization times and memory usage for TPC-H queries over PS-J

Context: PS-J scheme does not allow any partition pruning because join attributes do not occur in filter condition.

Result: Intermediate and advanced optimizers have less execution time than the basic optimizer. There is an average of 17% overhead from optimization time of these two optimizers than the basic one.

Results from DBA-Controlled Schemes



Execution times, optimization times and memory usage for TPC-H queries 5 and 8 over three partitioning schemes

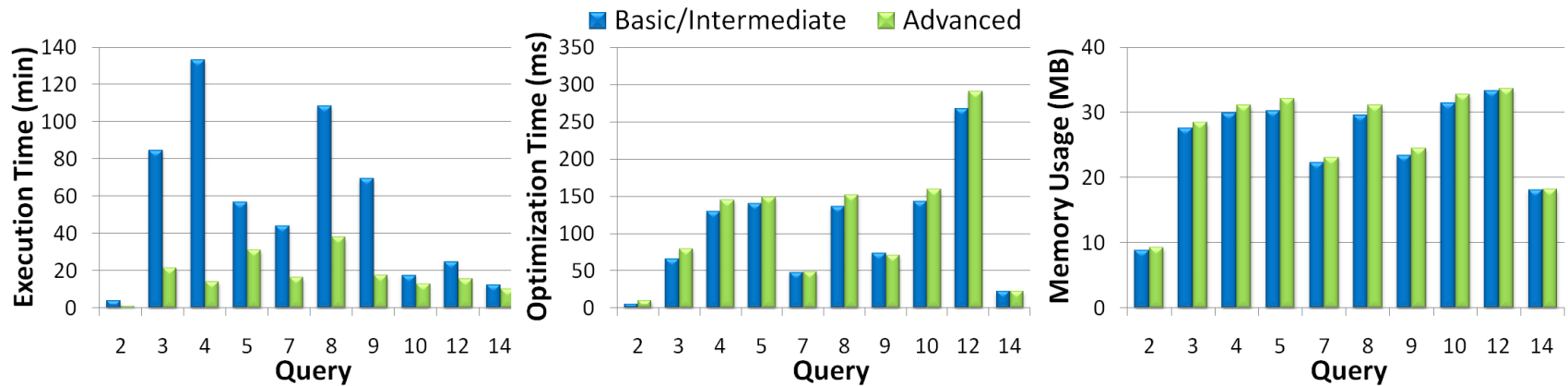
Context: PS-P only allows for partition pruning;

PS-J scheme prevents partition pruning since join attributes are not in filter condition;

PS-B scheme allows for both partition pruning and partition-wise joins.

Result: PS-P is best for query 8. PS-J is best for query 5. PS-B is best for both queries due to some workload or data properties.

Results from Constrained Schemes

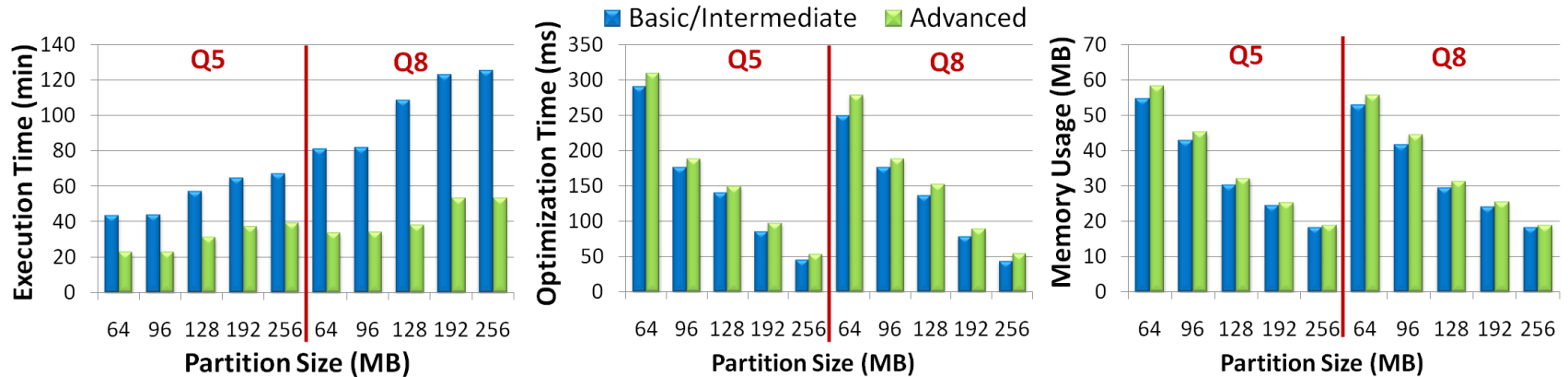


Execution times, optimization times and memory usage for TPC-H queries over PS-C partition size 128MB

Constraint: Not allow for one-to-one partition-wise joins

Result: Intermediate optimizer performs the same as basic one. Advanced optimizer provides over 2x speedup than the basic one for more than a half of the query. Overheads for opt. time and memory are 7.9% and 3.6% respectively.

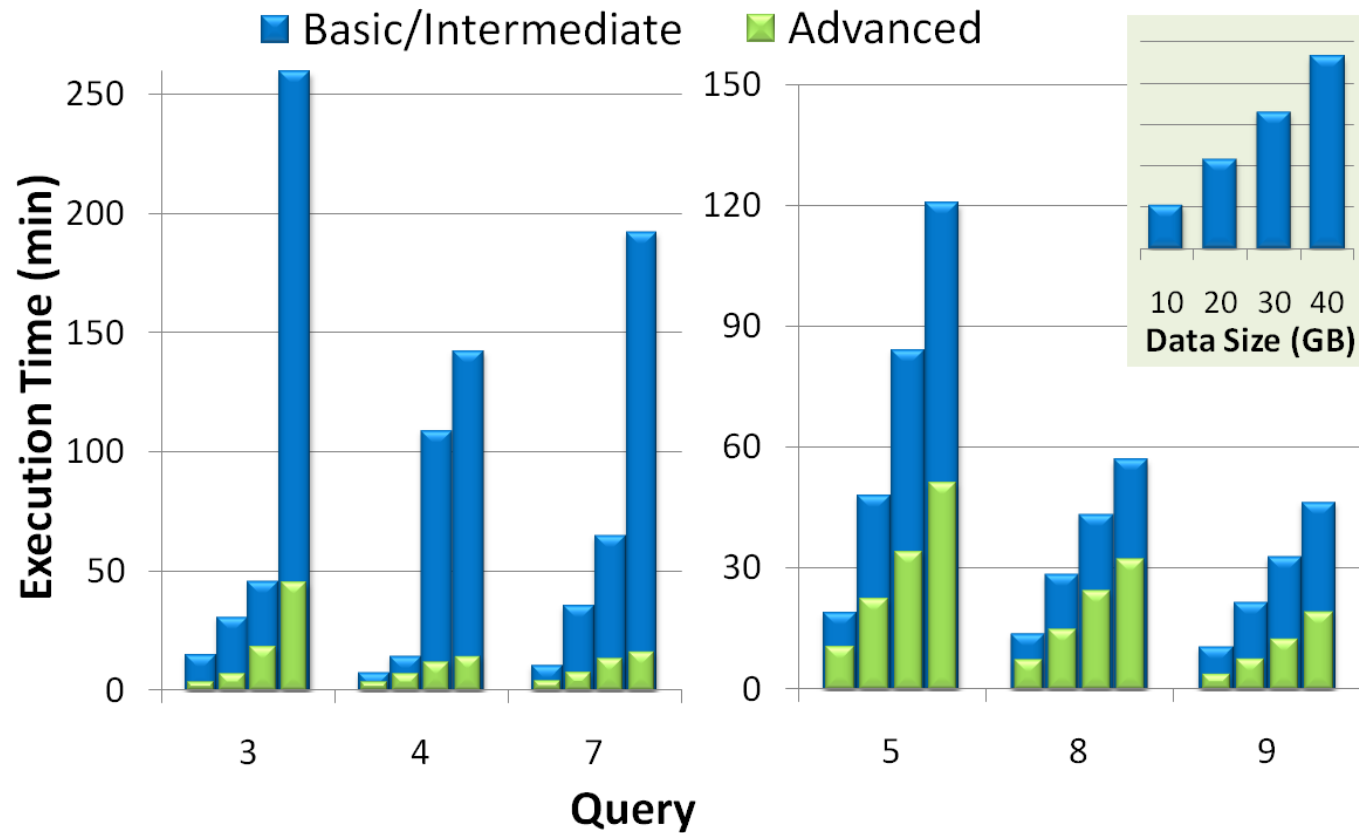
Effect of Size and Number of Partitions



Execution times, optimization times and memory usage for TPC-H queries 5 and 8 under PS-C scheme

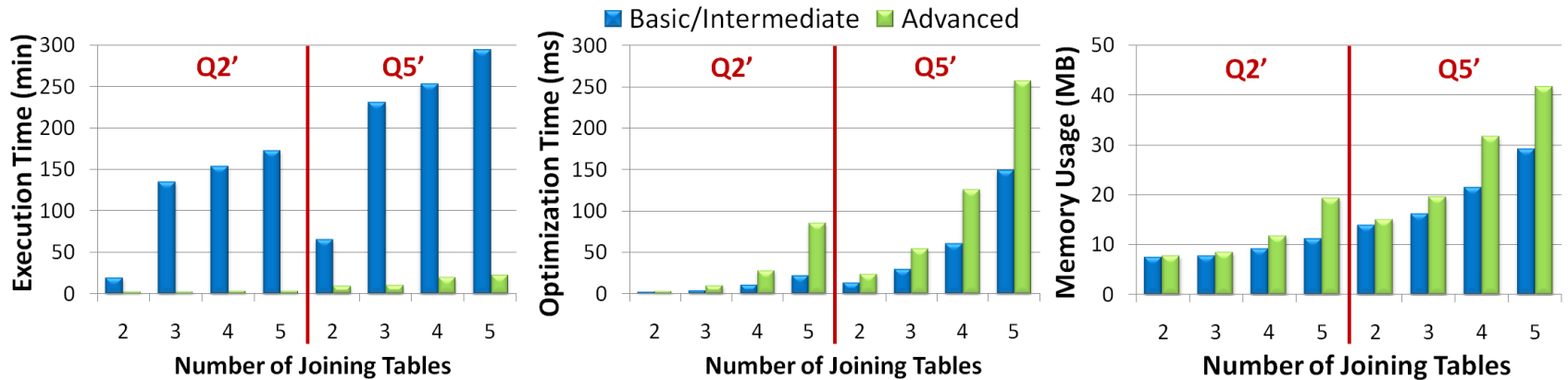
When partition size increases, the optimization time decreases for both optimizers. The smaller partitions are, the better partition pruning and join execution are due to finer-grained partition ranges. Therefore, the less execution time is obtained.

Effect of Data Size



Execution times VS. total data size under PS-C scheme

Stress Testing on a Synthetic Benchmark



Execution times, optimization times and memory usage change as the numbers of joining tables change.

Context: PS-C scheme; modified TPC-H queries 2 and 5 joining all the vertical tables for part and orders tables.

Result: the execution time of advanced optimizer has less impact on the number of joining tables, but its optimization time has larger impact on the amount of joining tables.

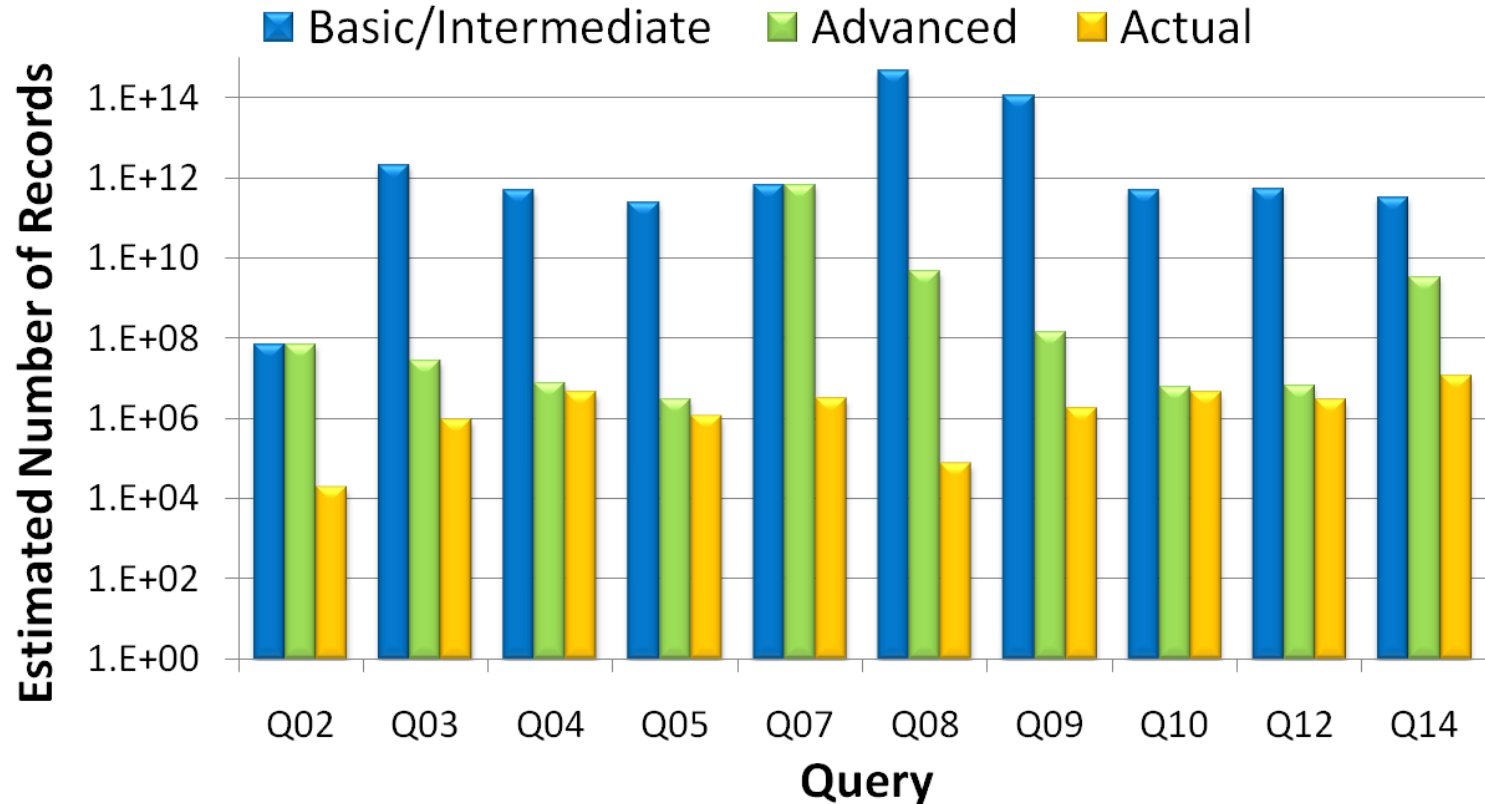
Effect of the Clustering Algorithm



Execution times, optimization times and memory usage with and without clustering under PS-C scheme

Result: Disabling clustering lead to high overhead for execution time, optimization time and memory usage. Execution time for certain queries not show in the figure is because of running out of memory during plan execution.

Results on Cardinality Estimation

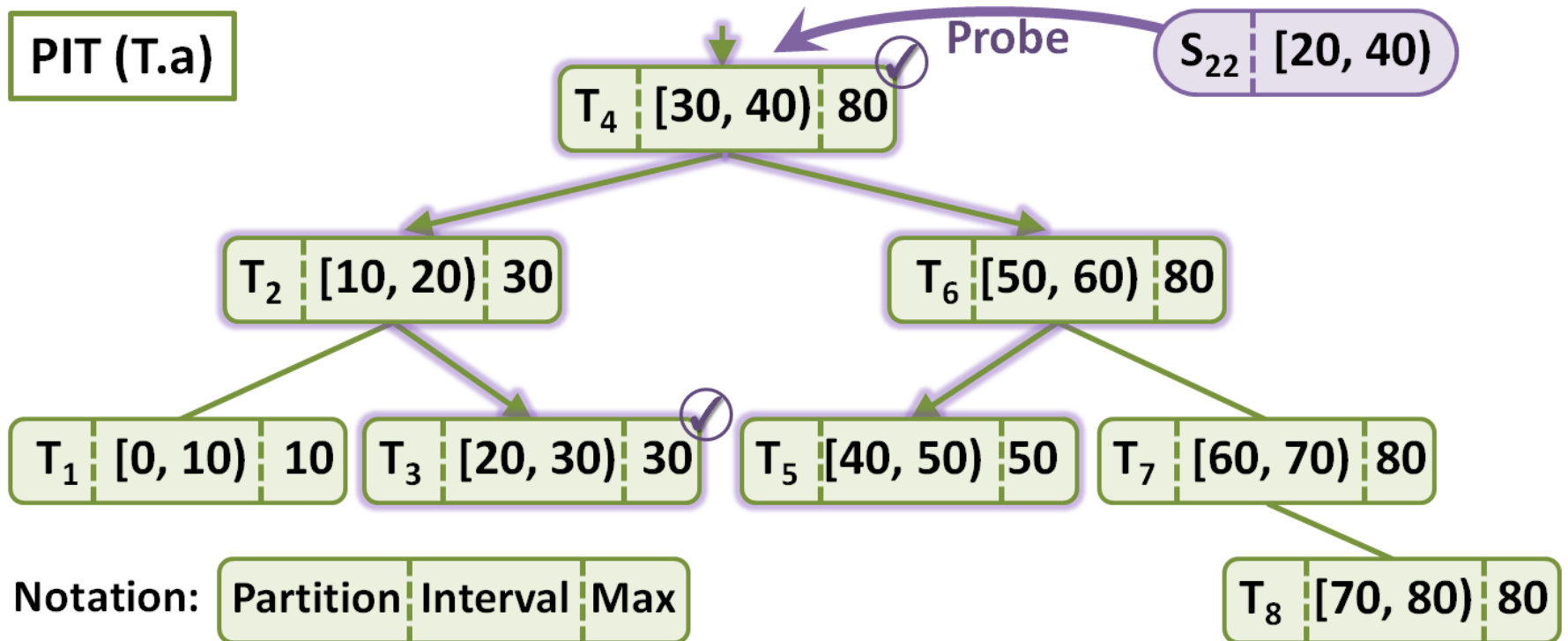


Estimated and actual number of records of TPC-H queries over PS-C scheme

Result: Advanced optimizer has better estimation since partition-wise joins increases the chances of using partition-level statistics directly for costing.

Critique and Discussion

- No detailed method is for probing the PIT tree.
Why S_{22} traverses 5 nodes instead of 3 nodes?



Critique and Discussion

- Details of TPC-H queries are not shared in this paper especially for query 3, 4 and 7.
- Experiments lack of IO activity results to demonstrate concurrent processing in experiments.
- It is semi-automatic optimizer due to involvement of DBA controls for partitioning schemes. (to optimize query performance to partition tables for optimizer)

Thank You.