Database Design

Wenfeng Xu
Hanxiang Zhao

Automated Partitioning Design in Parallel Database Systems

- MPP system:
- A distributed computer system which
- consists of many individual nodes, each of
- which is essentially an independent
- computer in itself.

- Bottelneck: Excessive data transfers
- How to cope?
- Originally partitioned in an adequate way

- Two categories:
- 1) Optimizer-independent
- 2) Shallowly-intergrated
- Two problems:
- 1) recommedations suffer from the tuning
- tools not being in-sync with optimizer's
- decisions
- 2)performance of the tuning tool is likely to
- dimish due to narrow APIs between the tool
- and the DBMS

- Advisor:
- Deeply-integrated
- Parallel query optimizer.

- PDW: appliance

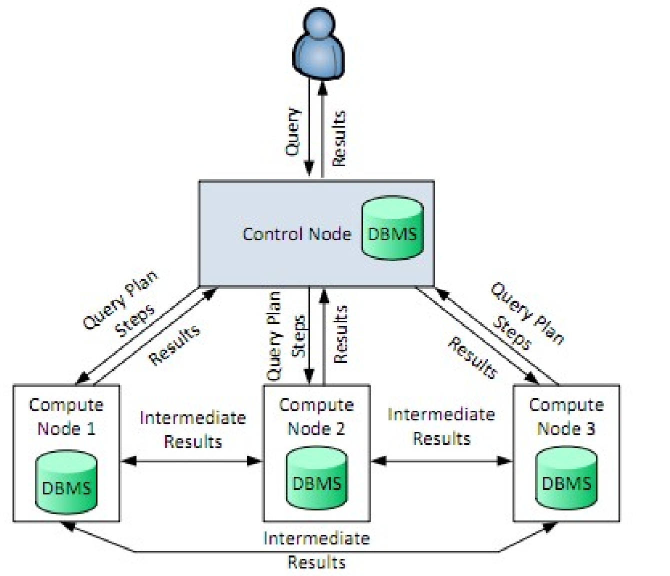


Figure 2: Overview of a 4 node appliance.

- Plan Generation and Execution

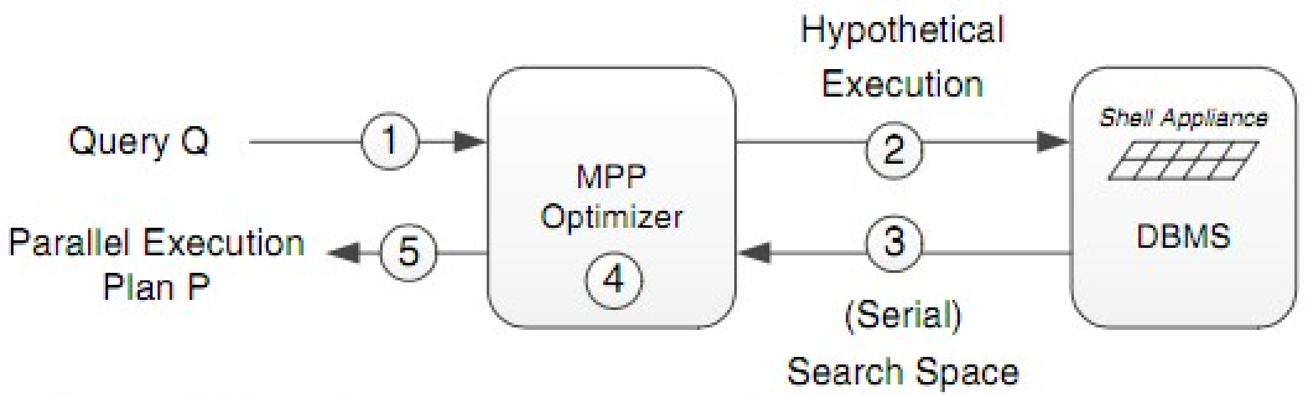


Figure 4: Parallel query optimization flow (all on the control node).

- Query plan->parallel execution plan(DSQL)
- DSQL:
- 1) SQL operations
- an SQL statement to be executed against
- the underlying compute node’s DBMS
- instance
- 2) Data movement operations
- transfer data between DBMS instances on
- different nodes

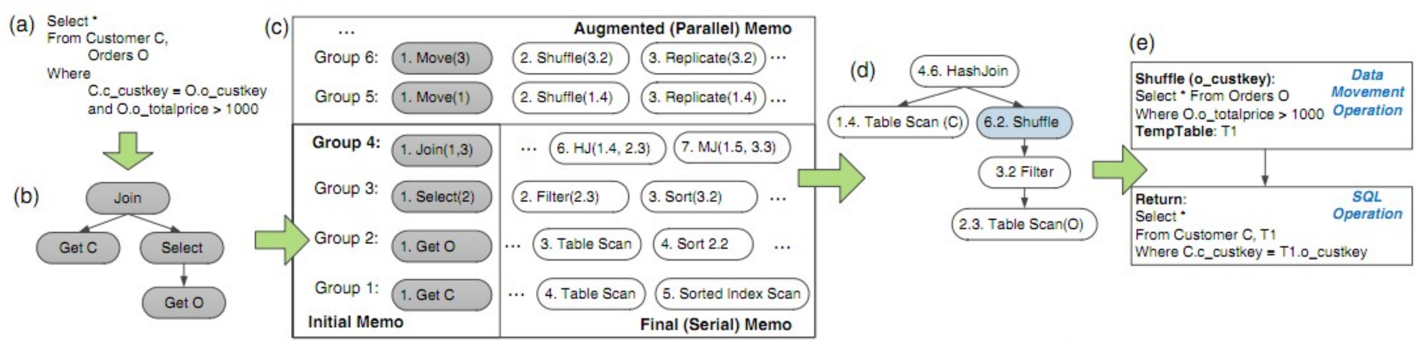


Figure 3: Parallel query optimization flow: (a) input query, (b) logical query tree, (c) augmented MEMO, (d) best query plan, (e) final DSQL plan.

Example: Consider the following SQL query:

```
SELECT *
FROM CUSTOMER C, ORDERS O
WHERE C.C_CUSTKEY = O.O_CUSTKEY
  AND O.O_TOTALPRICE > 1000
```

- MEMO: recursive data structure
- Groups and groupExpressions

- AUTOMATED PARTITIONING DESIGN
- PROBLEM
- *Given a database D, a query workload W,*
- *and a storage boundB, find a partitioning strategy (or configuration) for D such that*
- *(i) the size of replicated tables fits in B, and*
- *(ii) the overall cost of W is minimized.*

TUNING WITH SHALLOW OPTIMIZER INTERGRATION

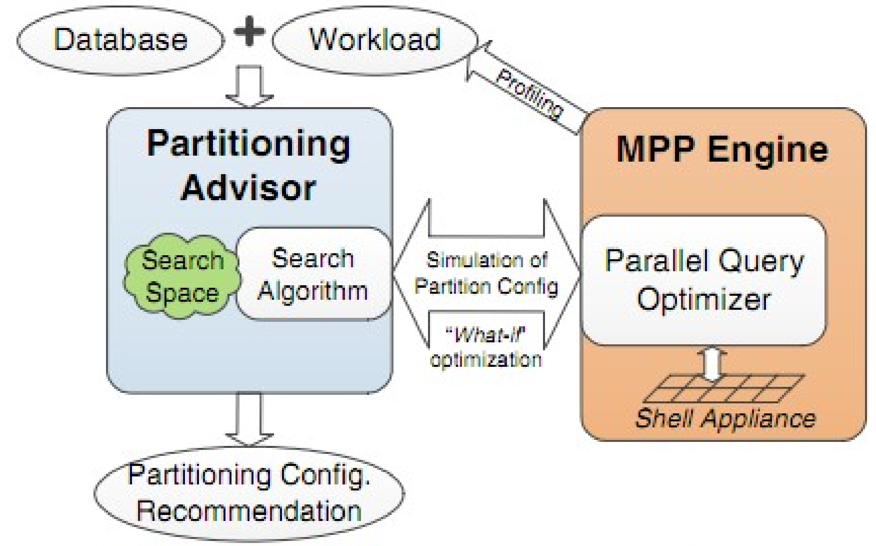


Figure 5: Partitioning advisor architecture.

- the complex search space
- the search algorithm
- the evaluation mechanism

- shallowly-integrated approach for
- partitioning tuning design:
- 1)Rank-Based Algorithm
- 2)Generic Algorithm

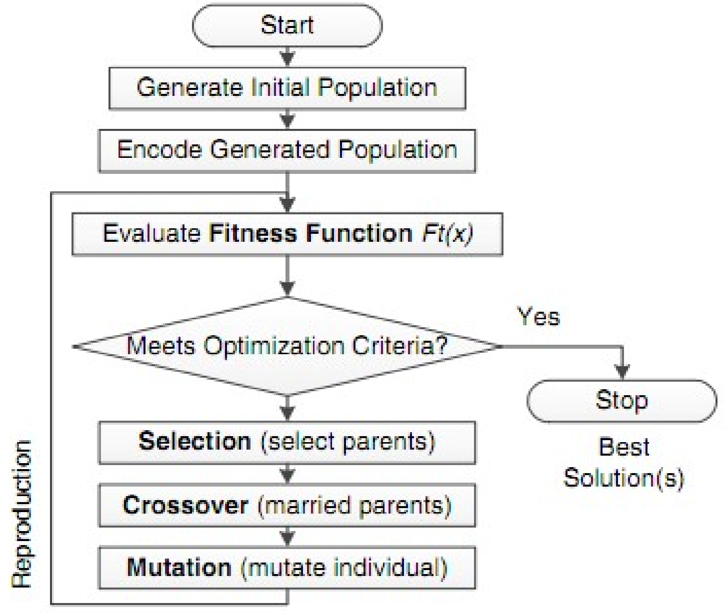


Figure 6: Flowchart for genetic algorithm.

- {nation, supplier, region, lineitem, orders,
- partsupp,
- customer, part} →
- {R,R,R,D1,D2,D1,D1,D1},

- Disadvantage of Shallowly-Integrated
- Approaches
- 1)search space is likely to be extremely
- large
- 2)each evaluation of a partitioning
- configuration is expensive

- TUNING WITH DEEP OPTIMIZER
- INTEGRATION
- MESA
- “workload memo”
- Figure 7:
- Interesting Columns
- 1)columns referenced in equality join
- predicates
- 2)any subset of group-by columns

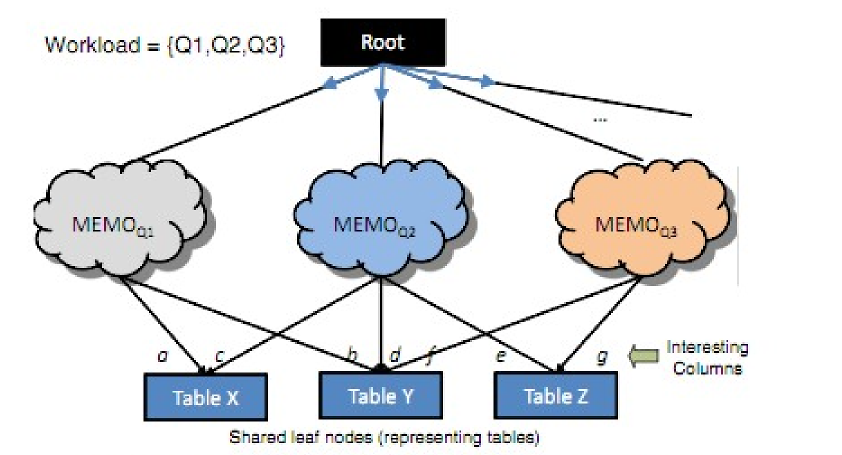


Figure 7: Workload MEMO data structure.

- *-partitioning:
- “every” partition or replication option for a
- base table is simultaneously available
- Branch and Bound Search
- Pruning:discards subtrees when a node or
- any of its descendants will never be either
- feasible or optimal

- Figure 8
- Node, Leaf, Bud, Bounding function,
- Incumbent
- 1)Node selection policy
- 2)Table/column selection policy
- 3)Pruning strategy
- 4)Bud node promotion
- 5)Stopping condition

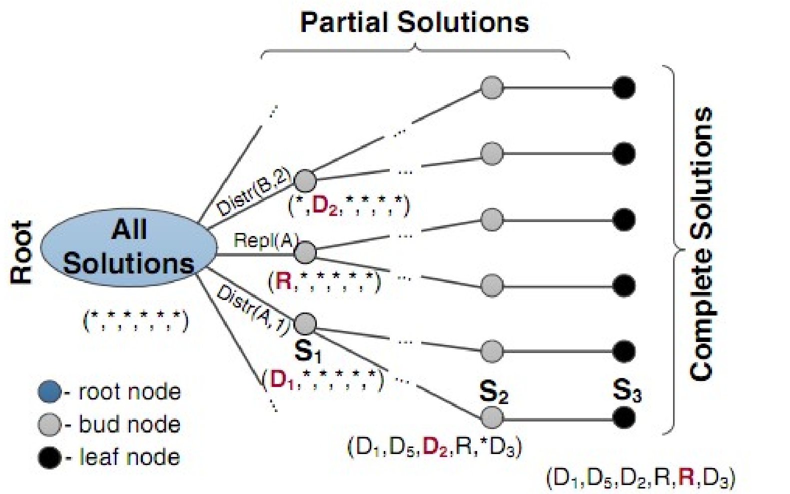


Figure 8: Branch and bound enumeration tree for partitioning configuration search problem.

MESA Algorithm

```
MESA (W:workload, B:storage bound)
wMemo = CreateWorkloadMemo(W, B)
incumbent = null
bbTree = CreateRoot(wMemo)
while (!stop_condition())
  currConfig = SelectNode(bbTree) // DFS policy
  newConfig = CreateChildConfig(currConfig) // table/column selection policy
  if (newConfig violates B constraint)
    prune(newConfig)
  else
    cost = ParallelPostProcess(wMemo, newConfig)
    if (newConfig is leaf or can be promoted)
      if (cost < incumbent.cost)
        incumbent = newConfig
      prune(newConfig)
    else // partially defined configuration
      if (incumbent.cost < cost)
        prune(newConfig)
return incumbent
```

Figure 9: Memo-based search algorithm using branch and bound enumeration.

- Experimental Evaluation
- Table 1,2,3
- We compare the quality of the
- recommendations produced by each
- technique

Benchmark (scale)	# Tables	Workload (# queries)
TPC-H (1TB)	8	22
TPC-DS (1TB)	25	50
L'Oreal (88GB)	573	29
MSSales (800GB)	346	27

Table 1: Experimental benchmarks

Parameter	Value	Description
# of generations	100	# of times the population will be replaced through reproduction.
Population size	30	# of chromosomes available for use during the search. If the size is too big, GA will spend unnecessarily long time evaluating chromosomes, if it is too small, GA may have no chance to adequately cover the search space.
Crossover rate	0.1	the probability of crossover between two chromosomes.
Mutation rate	0.1	the probability that values of genes of a newly created (or selected) off-springs will be randomly changed.
Selection rate	0.2	the percentage of the worst of the current population that will be discarded (after re-generation)

Table 2: GA parameters

Parameter	Value	Description
Node selection	DFS	the forward- and the back-tracking policy in the branch and bound tree
Variable selection	replicate, distribute by rank	See Section 5.5 for details.
Stop condition	150	the number of iterations after which the search terminates

Table 3: MESA parameters

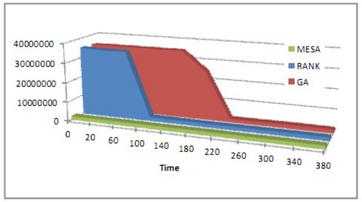


Figure 10: Quality over time: TPC-H.

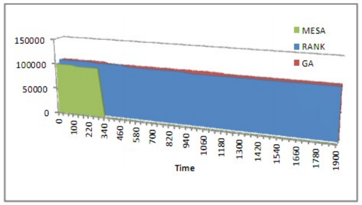


Figure 12: Quality over time: L'Oreal.

	Approach	Quality(Q) Visual Depict.	MESA Imp(Q)	Total Time(T)	MESA Imp(T)
TPC-H	RANK		1.1x	6 min 42 sec	1.73x
	GA		1.3x	4 hrs 28 min	69.3x
	MESA		–	3 min 52 sec	–
TPC-DS	RANK		1.1x	3 h 53 min	7.13x
	GA		1.6x	25 h 28 min	46.7x
	MESA		–	32 min 42 sec	–
L'Oreal	RANK		1.2x	45 min 25 sec	9.05x
	GA		1.2x	14 h 12 min	169x
	MESA		–	5 min 1 sec	–
MSSales	RANK		1.1x	6 h 11 min	3.37x
	GA		1.2x	27 h 39 min	15.1x
	MESA		–	1 h 50 min	–

Table 4: Comparison of techniques

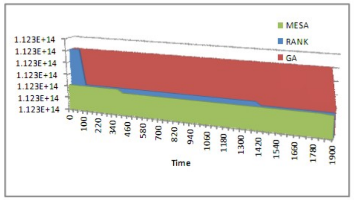


Figure 11: Quality over time: TPC-DS.

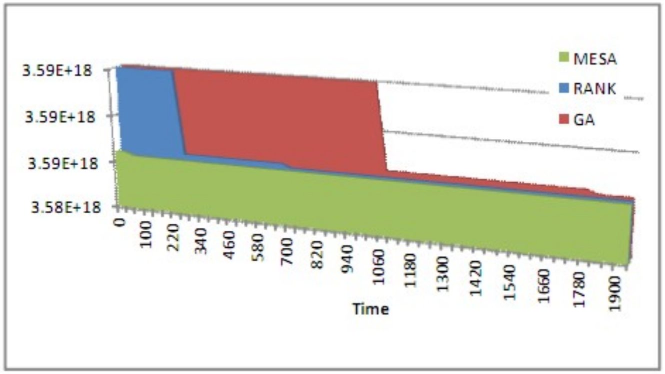


Figure 13: Quality over time: MSSales.

Impact of replication bound

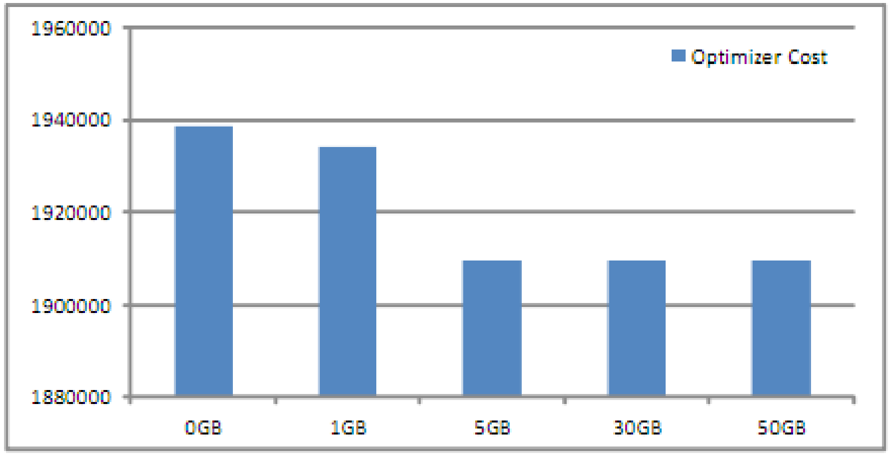


Figure 14: Quality of recommendations under various replication bounds.

- Performance of MESA
- Workload MEMO construction overhead

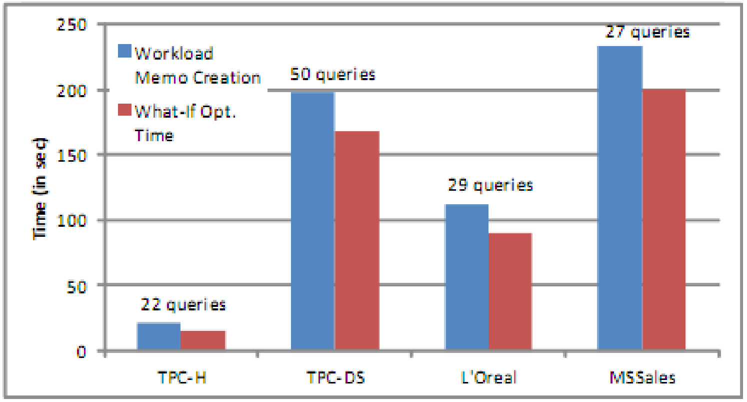


Figure 15: Time overhead of workload MEMO creation.

- Subsequent reoptimization calls

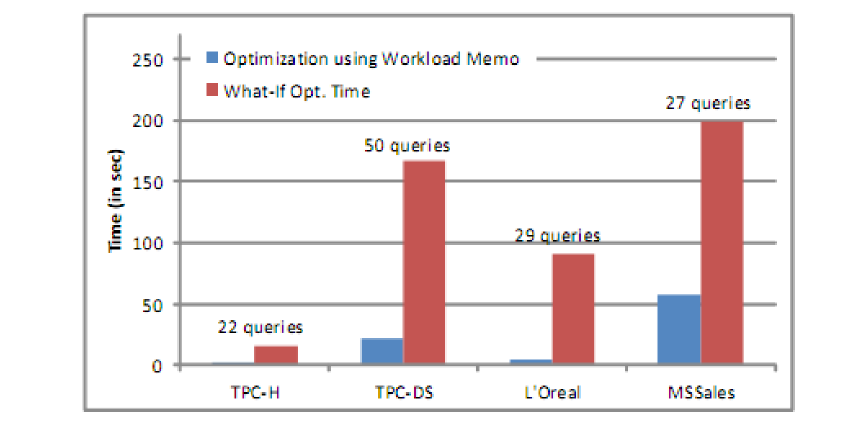


Figure 16: Speedup of subsequent optimizations using workload MEMO.

- EXTENSIONS
- Updates
- Multi-Column Partitioning
- Range Partitioning
- Interaction With Other Physical Design
- Structures

- CONCLUSION
- techniques for finding the best partitioning
- configuration in distributed environments
- deep integration with the parallel query
- optimizer
- Using its internal MEMO data structure for
- faster evaluation of partitioning
- configurations and to provide lower bounds
- during a branch and bound search strategy

Schism: a Workload-Driven Approach to Database Replication and Partitioning

Background

- Problem:

  distributed transactions are expensive in OLTP settings.

  why: two-phase commit

- Solution:

  minimize the number of distributed transactions, while producing balanced partitions.

- Introduce:

  Schism

  H-store

Schism

- Five steps:
- Data pre-procession
- Creating the graph
- Partitioning the graph
- Explaning the partition
- Final validation

Graph Representation

- notion: node, edge, edge weights
- example: a bank database (from paper)
- workload: 4 transactions

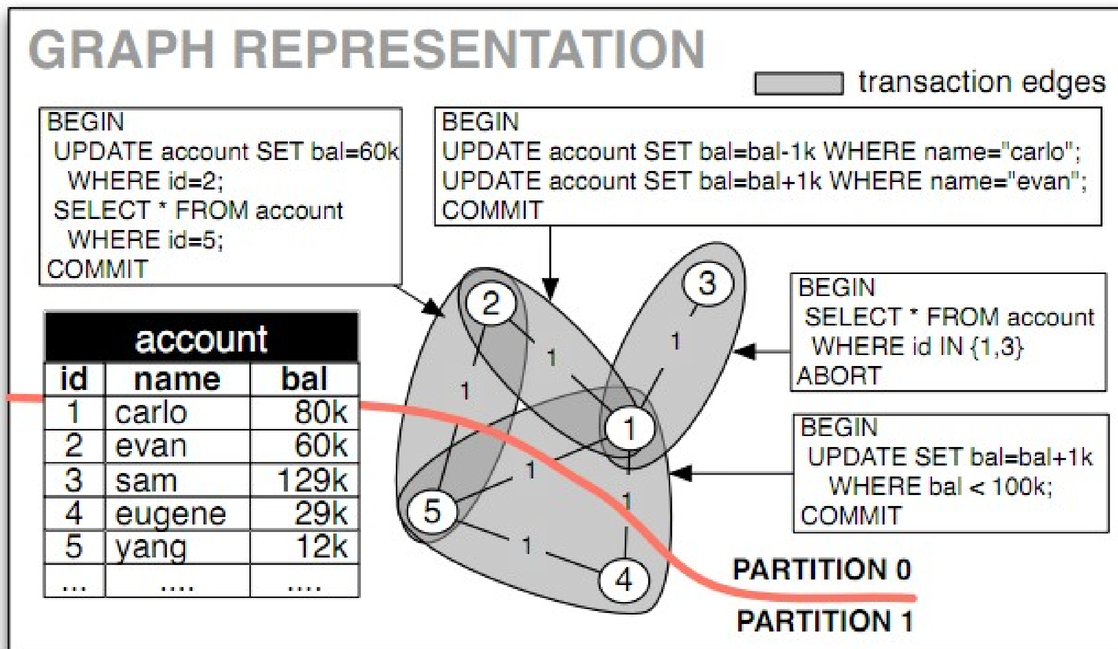


Figure 2: The graph representation

Graph Representation

- an extension of the basic graph representation
- Graph replication: “exploding” the node representing a single tuple into a star-shaped configuration of n + 1 nodes. （Figure 3 from paper)

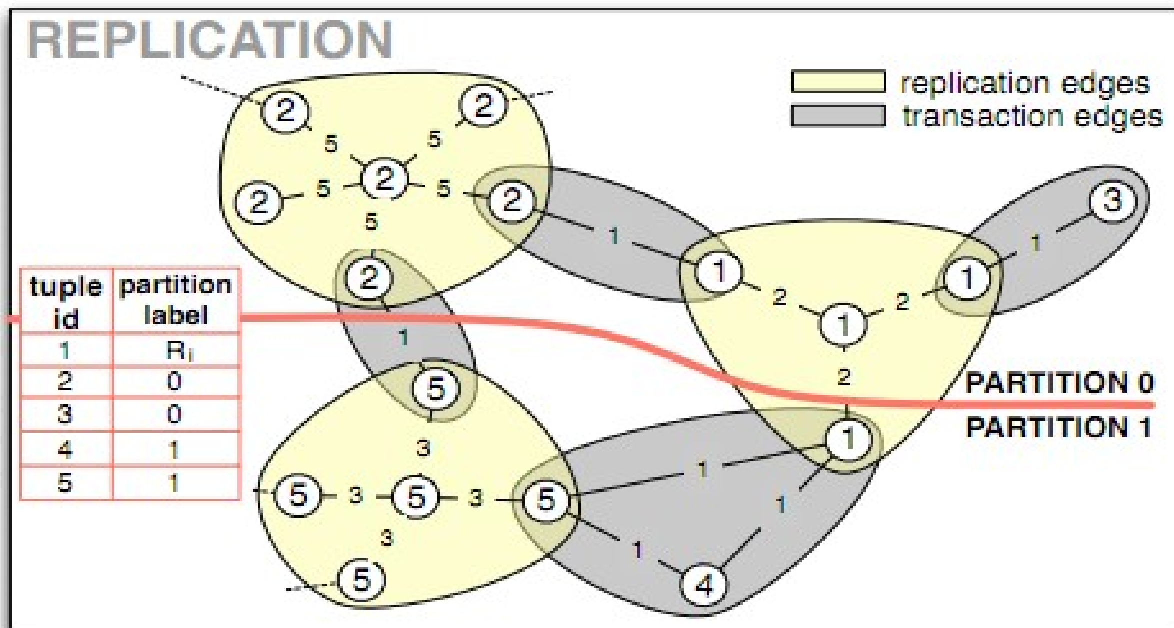


Figure 3: Graph with replication

Graph Partitioning

- split graph into k partitions→overall cost of the cut edges is minimized.
- result: a fine-grained partition
- lookup table: node--partition label
- note: replicated tuple

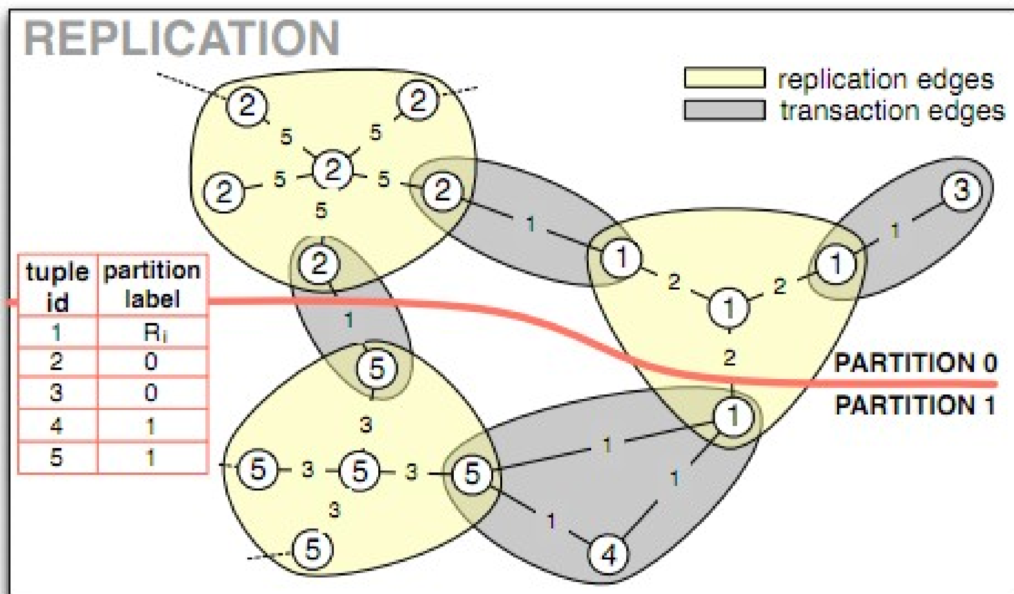


Figure 3: Graph with replication

Explanation Phase

- use decision tree to find a compact model that captures the (tuple, partition) mappings.
- (id = 1) → partitions = {0, 1}
- (2 ≤ id < 4) → partition = 0
- (id ≥ 4) → partition = 1

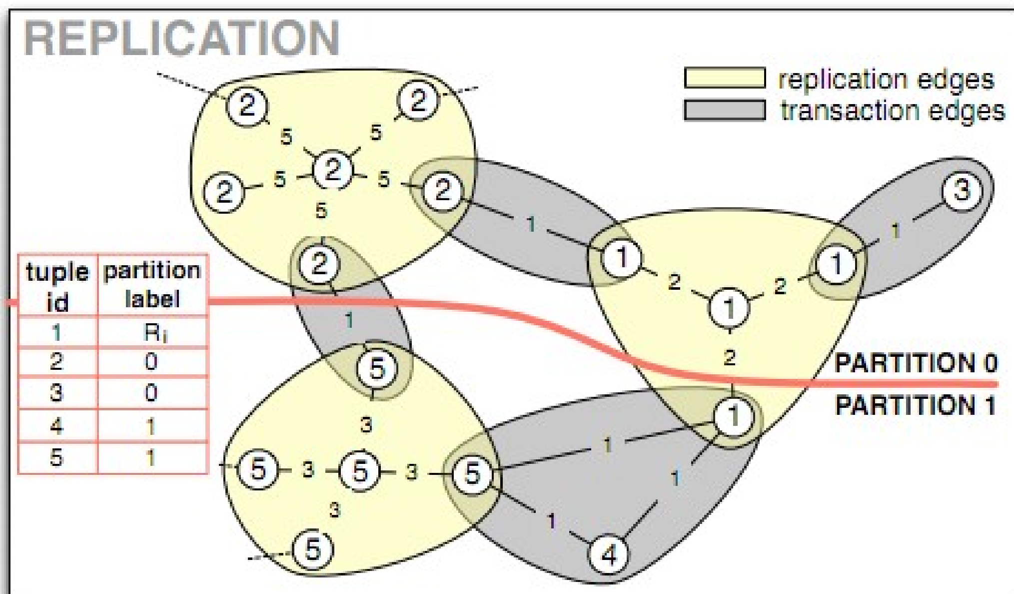


Figure 3: Graph with replication

Final Validation

- compare solutions to select the final partitioning scheme.
- fine-grained per-tuple partitioning,range-predicate partitioning, hash-partitioning

Optimization

- graph partitioners scale well in terms of the number of partitions, but running time increases substantially with graph size.

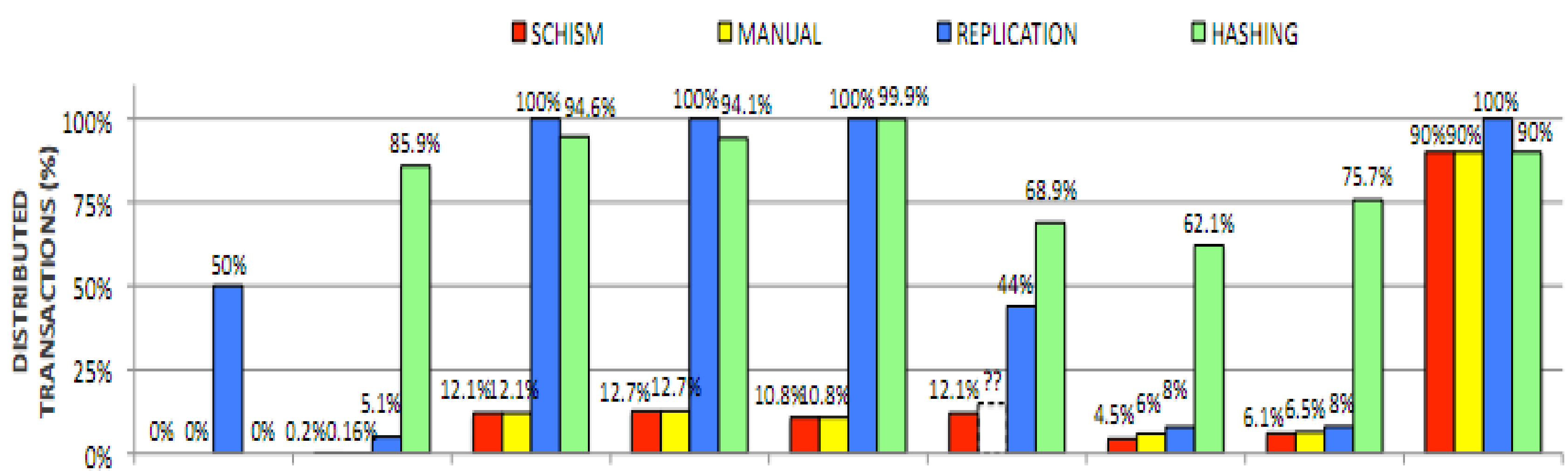
- methods for reducing size of graph:

  transaction-level sampling

  tuple-level sampling

  tuple-coalescing

Experimental Evaluation



dataset:	YCSB-A	YCSB-E	TPCC-2W	TPCC-2W	TPCC-50W	TPC-E	EPINIONS	EPINIONS	RANDOM
partitions:	any	100	2	2	10	2	2	10	10
coverage:	any	1%	50%	0.5%	1%	5%	15%	15%	any
SCHISM:	hashing	range-predicates	range-predicates	range-predicates	range-predicates	range-predicates	look-up table	look-up table	hashing

Figure 4: Schism database partitioning performance.

Thank you!