Incremental Query Optimization

Vipul Venkataraman

Dr. S. Sudarshan

Computer Science and Engineering
Indian Institute of Technology Bombay
Outline

Introduction
  Volcano
  Cascades

Incremental Optimization
  Priority Metrics
  Deferred Cost Propagation

Experiments
  Current Best Cost Plots
  Area Under Curve Plot
  Initial Plan

Conclusion

Future Work

References
Introduction

Volcano
Cascades

Incremental Optimization
Priority Metrics
Deferred Cost Propagation

Experiments
Current Best Cost Plots
Area Under Curve Plot
Initial Plan

Conclusion

Future Work

References
Active area of research - spanning more than four decades [1]
Determine the most efficient way to execute a given query by considering the space of all possible query plans
The optimization process should not be costly in itself - typical issue in web related applications that handle huge data
More data \(\Rightarrow\) bigger search space
Can we find a good enough plan without exploring the whole search space?
Logical Query DAG

Logical Query DAG for the query $A \Join B \Join C$

Image source: Prasan Roy's thesis
Physical Query DAG for the query $A \bowtie B$

Image source: Prasan Roy’s thesis
Steps in Optimization

- Incremental Query Optimization

Introduction

Image source: Prasan Roy’s thesis
Volcano, Graefe [2]

- Top-down optimizer, uses dynamic programming and memoization
- Expands whole Logical Query DAG initially to create all possible logical expressions
- Enumerates physical search space in depth-first order
- Uses branch and bound pruning to prune the search space
- Cost upper bound is propagated down the DAG, if upper bound < lower bound, plan is pruned
Cascades, Graefe et al [3]

- Adheres to the Object Oriented Paradigm
- Optimization algorithm broken into task objects which are collected in a data structure
- A task is scheduled by popping it from the stack and invoking the perform function
- Only explores a group on demand, while Volcano exhaustively generates all logically equivalent expressions before the actual optimization begins
- Ordering of transformations to be applied is based on a promise value assigned to each task
Based on the Cascades optimizer generator

Lower bound group pruning

- A group lower bound associated with a group of logically equivalent expressions, represents the minimal cost of the group
- Based on the logical properties of the group

Global epsilon pruning

- A plan is considered optimal plan for an equivalence node if its cost is close enough (within epsilon) to the cost of optimal plan
- If the cost of a plan is found less than epsilon, it is good enough
- Epsilon values affect pruning greatly
Based on the Cascades optimizer generator

Lower bound group pruning

- A group lower bound associated with a group of logically equivalent expressions, represents the minimal cost of the group
- Based on the logical properties of the group

Global epsilon pruning

- A plan is considered optimal plan for an equivalence node if its cost is close enough (within epsilon) to the cost of optimal plan
- If the cost of a plan is found less than epsilon, it is good enough
- Epsilon values affect pruning greatly
Based on the Cascades optimizer generator

**Lower bound group pruning**

- A group lower bound associated with a group of logically equivalent expressions, represents the minimal cost of the group
- Based on the logical properties of the group

**Global epsilon pruning**

- A plan is considered optimal plan for an equivalence node if its cost is close enough (within epsilon) to the cost of optimal plan
- If the cost of a plan is found less than epsilon, it is good enough
- Epsilon values affect pruning greatly
Limitations of The Cascades Framework

Data model representation is tightly coupled with optimizer

- Requires database implementer to define logical & physical operators and logical & physical properties
- Difficult to integrate a new query optimizer to an existing database, as it requires a database implementer to write new class definitions [5]

Depth-first search algorithm

- Implements depth-first search algorithm; no plan generated unless whole optimization process finishes
- Huge search space of query optimizer leads to inefficiency
Limitations of The Cascades Framework

Data model representation is tightly coupled with optimizer

- Requires database implementer to define logical & physical operators and logical & physical properties
- Difficult to integrate a new query optimizer to an existing database, as it requires a database implementer to write new class definitions [5]

Depth-first search algorithm

- Implements depth-first search algorithm; no plan generated unless whole optimization process finishes
- Huge search space of query optimizer leads to inefficiency
Ideas for improvements

- The current best plan can be used in the search algorithm to enable efficient exploration.
- This will enable incremental view maintenance.
- Depth first search can be replaced with a more efficient *Best first search* strategy.
- Tasks can be operated depending on their *promise*.
- Pruning can be done more aggressively, at the risk of a near optimal plan.
Introduction
Volcano
Cascades

**Incremental Optimization**
Priority Metrics
Deferred Cost Propagation

Experiments
Current Best Cost Plots
Area Under Curve Plot
Initial Plan

Conclusion

Future Work

References
Incremental Optimization

Maintaining current best plan

- The optimal plan for a query is converged upon by incrementally improving a current best plan through application of logical and physical transformations
- At each step of the optimization, a *current best plan* is available, which is the plan with the best cost among all plans that have been explored
- The order of exploring alternative plans becomes important as it controls the time required to reach the optimal plan, and the amount of search space that can be pruned

Benefits

- Pruning can be done aggressively, at the risk of returning a near-optimal plan
- Optimization can be aborted if the current best plan cost reaches a certain threshold, allowing for quickly returning near-optimal plans
Incremental Optimization

Maintaining current best plan

- The optimal plan for a query is converged upon by incrementally improving a current best plan through application of logical and physical transformations.
- At each step of the optimization, a current best plan is available, which is the plan with the best cost among all plans that have been explored.
- The order of exploring alternative plans becomes important as it controls the time required to reach the optimal plan, and the amount of search space that can be pruned.

Benefits

- Pruning can be done aggressively, at the risk of returning a near-optimal plan.
- Optimization can be aborted if the current best plan cost reaches a certain threshold, allowing for quickly returning near-optimal plans.
Incremental Optimization: Issues

Prioritization of Transformations

- The performance of the incremental optimizer is greatly affected by the order in which the search space is explored.
- Enumeration of search space is done by applying transformations at each operator in the logical plan.
- At each step, multiple valid transformations may exist.
- If promising plans are expanded first, the less promising ones will be pruned and not expanded at all.
- Difficult to ascertain which plans are promising, before expanding them.
- Optimal selection depends on various factors, e.g. the type of operator being substituted, the transformation rule, logical and physical properties of the required context, database statistics, etc.
Heuristic Ordering of Nodes

- The order in which nodes are selected for optimization determines the performance of the incremental optimizer.
- But the best order depends on the query as well as dataset characteristics.
- Developing a set of heuristics for choosing the order of exploration of nodes, that take into account the operator, child nodes, logical and physical properties of the query, should improve the performance of the optimizer.
- Converging on a set of good heuristics would require an analysis of the performance of the optimizer under various heuristics.
Priority Metrics

- Depth from the root of the DAG
- Input Blocks
- Cost upper bound
Priority Metrics

Depth from the root of the DAG

- Length of the path from the root of the DAG to that particular node
- Depth assigned to explore the DAG in depth-first-search fashion
- Priority assigned during exploration:
  - root.priority = 0
  - child.priority = root.priority + 1;
Priority Metrics

Input Blocks Size

- For an equivalence node, it is equal to the number of blocks in which it will reside in memory
  
  \[ \text{inputblocks(equivalence node)} = \frac{\text{number of tuples}}{\text{number of tuples per block}} \]

- For an operator node, it is equal to the number of blocks that need to be read to perform that operation
  
  \[ \text{inputblocks(operator node)} = \sum \text{inputblocks(children equivalence node of that operator node)} \]

* this approximation does not hold in the case of nested loop joins. We plan to reformulate the cost model soon
Priority Metrics

Cost Maintenance

- Cost of a node is equal to the best plan cost to evaluate that node
- Any new plan obtained at a node results in propagating it up DAG to modify the best plan costs of the parent nodes
- Cost computations is done as:
  - *Operator Node*: The cost of an operator node is the sum of costs of all its children equivalence nodes in addition to its local cost
  - *Equivalence Node*: The cost of an equivalence node is the minimum of the costs of its children operator nodes
Upper Bound Cost Maintenance

- After updating the best plan cost for the required nodes, upper bound is updated for all the children recursively in top down fashion.
- Upper Bound Cost computations is done as:
  - **Operator Node**: The upper bound of an operator node is the upper bound of its parent equivalence node.
  - **Equivalence Node**: Consider a hierarchy of nodes like e1-p1-e2. Upper bound of e2 is calculated as:

\[
U_{e2} = U_{e1} - \sum_{i \in \text{siblings}(e2)} L_i - \text{Local}_{p1}
\]

where \( U_x \) is the upper bound cost of \( x \), \( L_x \) is the lower bound cost of \( x \).
Deferred cost propagation

- Whenever cost changes at a node, it results in updating best plan cost at all parents and upper bounds at all children.
- Updates can be ignored if change in best plan cost is negligible at the node.
- Following things are expected if revisiting large portions of the graph due to small changes in local cost is avoided:
  - *Faster optimization*: Faster computation of the plan is expected since graph exploration is avoided when not necessary.
  - *Sub-optimal plans*: Sub-optimal plan is expected since all changes in the cost are not updated.
Deferred Cost Propagation

Fudge factor

- Change is propagated only when the cost at a node differs at least by a factor \( \alpha \) - fudge factor
- The fudge factor will essentially balance the trade-off between faster optimization and obtaining near-optimal plans
- Fudge factor needs to be tuned to ensure:
  - The time taken for optimization is faster
  - A near-optimal plan is generated
Introduction
  Volcano
  Cascades

Incremental Optimization
  Priority Metrics
  Deferred Cost Propagation

Experiments
  Current Best Cost Plots
  Area Under Curve Plot
  Initial Plan

Conclusion

Future Work

References
Implementation

- PyroJ optimizer [6]
- Evaluate quality of various search strategies

Heuristics

1. Depth first search
2. Maximum cost upper bound
3. Maximum input blocks
4. Minimum cost upper bound
5. Minimum input blocks
Implementation

- PyroJ optimizer [6]
- Evaluate quality of various search strategies

Heuristics

1. Depth first search
2. Maximum cost upper bound
3. Maximum input blocks
4. Minimum cost upper bound
5. Minimum input blocks
Tracking improvements in current best cost

- We present a series of experiments measuring the changes in the current best cost plan as the query optimization process progresses.
- The 5 implemented heuristics are evaluated on a fixed set of 10 queries on the TPC-DS dataset.

Current best cost plot

- x axis shows the time in milliseconds.
- y axis shows the cost of the current best plan, which improves until it reaches the optimal cost.
- Each plot shows the results for a fixed query, under different heuristics.
- There is a curve for each heuristic under consideration.
Tracking improvements in current best cost

- We present a series of experiments measuring the changes in the current best cost plan as the query optimization process progresses.
- The 5 implemented heuristics are evaluated on a fixed set of 10 queries on the TPC-DS dataset.

Current best cost plot

- x axis shows the time in milliseconds.
- y axis shows the cost of the current best plan, which improves until it reaches the optimal cost.
- Each plot shows the results for a fixed query, under different heuristics.
- There is a curve for each heuristic under consideration.
Current Best Cost Plots

Incremental Query Optimization Experiments
Current Best Cost Plots

Incremental Query Optimization
Experiments
Selecting the best heuristic

- The heuristic has a reasonable effect on the performance of the optimizer
- We would prefer a heuristics that provides optimal (or near-optimal) plans in as few iterations as possible
- Next slide shows a plot of the area-under-curve of each heuristic under consideration

Area under curve plot

- There is a curve for each heuristic in the plot
- A heuristic with a low AOC value for a particular query, relative to other heuristic, implies that that heuristic is a better choice for that query
- We are interested in heuristics that perform consistently well on all queries
Selecting the best heuristic

- The heuristic has a reasonable effect on the performance of the optimizer
- We would prefer a heuristics that provides optimal (or near-optimal) plans in as few iterations as possible
- Next slide shows a plot of the area-under-curve of each heuristic under consideration

Area under curve plot

- There is a curve for each heuristic in the plot
- A heuristic with a low AOC value for a particular query, relative to other heuristic, implies that that heuristic is a better choice for that query
- We are interested in heuristics that perform consistently well on all queries
Area Under Curve Plot
Initial plan

Importance

- We have a plan to work with
- Incremental view maintenance

Algorithm

- *Operator Node*: add all children equivalence nodes to the initial plan
- *Equivalence Node*: add any *one* child (operator node) to the initial plan
Initial plan

Importance

- We have a plan to work with
- Incremental view maintenance

Algorithm

- *Operator Node*: add all children equivalence nodes to the initial plan
- *Equivalence Node*: add any *one* child (operator node) to the initial plan
Area Under Curve Plot
Effect of using an initial plan

- Observable difference among different heuristics
- Using initial plan improved performance 😊
- Minimum cost upper bound heuristic + initial plan is the winner
Deferred Cost Propagation

- Fudge factors used
  - 0.00 - returns best plan!
  - 0.10
  - 0.15
  - 0.20
- We hope that
  - increasing fudge factor reduces optimization time
  - we obtain good sub-optimal plans
- This indeed happens 😊
Deferred Cost Propagation

![Graph showing query time vs. query number with different cost propagation values]
Deferred Cost Propagation

![Graph showing percentage change in cost vs query with markers for different values.]
Introduction
  Volcano
  Cascades

Incremental Optimization
  Priority Metrics
  Deferred Cost Propagation

Experiments
  Current Best Cost Plots
  Area Under Curve Plot
  Initial Plan

Conclusion

Future Work

References
We presented a literature survey in the area of traditional database optimization techniques.

We explained the incremental optimization technique in the context of query optimization.

The overhead of storing the current best plan in incremental optimization can be offset by intelligently exploring the plan space.

We achieved this using various heuristics that lead to faster discovery of optimal or near-optimal plans, as well as greater pruning of the plan space.

We have implemented deferred cost propagation coupled with heuristic search.

Our experimental results show the significant benefits of our techniques.
Introduction
Volcano
Cascades

Incremental Optimization
Priority Metrics
Deferred Cost Propagation

Experiments
Current Best Cost Plots
Area Under Curve Plot
Initial Plan

Conclusion

Future Work

References
Future Work

- Effective initial plan computation methods
- Theoretical guarantees for heuristic performance
- Adaptive heuristic optimization
- Global epsilon pruning
- Evaluating on different dataset/queries
Introduction
  Volcano
  Cascades

Incremental Optimization
  Priority Metrics
  Deferred Cost Propagation

Experiments
  Current Best Cost Plots
  Area Under Curve Plot
  Initial Plan

Conclusion

Future Work

References
References

An Overview of Query Optimization.

The Volcano optimizer generator: extensibility and efficient search.  

The cascades framework for query optimization.  

Efficiency in the columbia database query optimizer.  

Query optimization for massively parallel data processing.  
Thank you 😊