Memory Cognizant Query Optimization

Arvind Hulgeri
Krithi Ramamritham
S. Seshadri
S. Sudarshan

Department of Computer Science and Engineering
I.I.T. Bombay
May 1999

Why Memory Cognizant Optimizer?

Conventional Optimizer

- Memory intensive operators.
- Typical optimizers assume all the memory to be available to each operator in the query tree: *Wrong assumption.*
- Memory will get divided amongst all the operators running simultaneously in a pipeline.
- How to divide the memory optimally amongst the operators of a pipeline?

2-Phase Approach *v/s* 1-Phase Approach

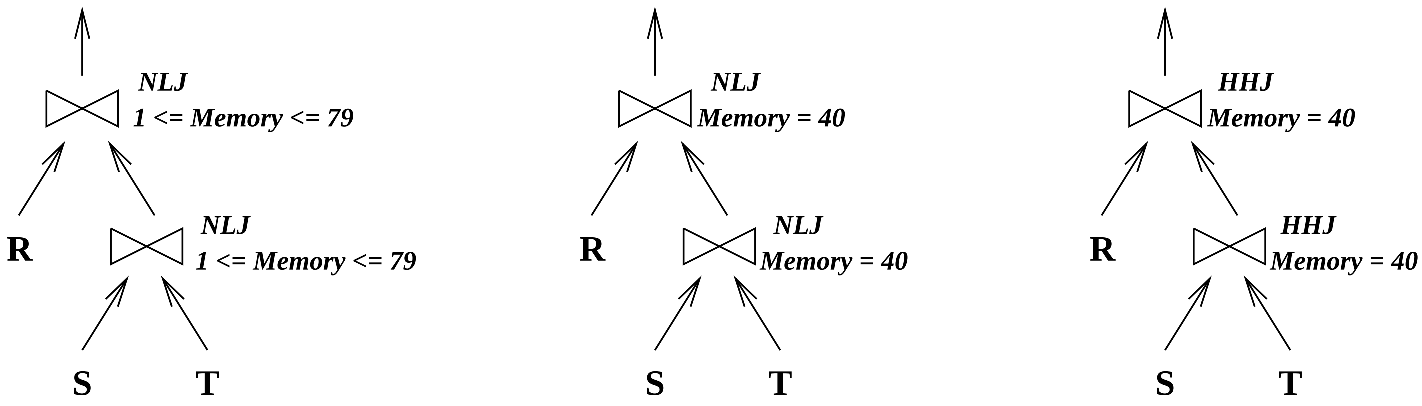
2-Phase Approach

- Optimize the query using conventional optimizer.
- Distribute available memory optimally amongst the operators in a pipeline.

Problem: May not give optimal execution time as the plan generated may itself be sub-optimal for the given memory.

1-Phase Approach

In this approach, we propose to modify the traditional optimizer to make it memory cognizant. Plan chosen takes into account the division of memory amongst operators. *Motivation...*



(a) Conventional Optimizer	(b) 2-Phase Optimizer	(c) 1-Phase Optimizer
Estimated Plan Cost (IO) = 240	Estimated Plan Cost (IO) = 610	Estimated Plan Cost (IO) = 500
Actual Plan Cost (IO) = [610...7860]	Actual Plan Cost (IO) = 610	Actual Plan Cost (IO) = 500

|R| = 60, |S| = 60, |T| = 130 Memory = 80

Why Memory Cognizant Optimizer?: A Motivating Example

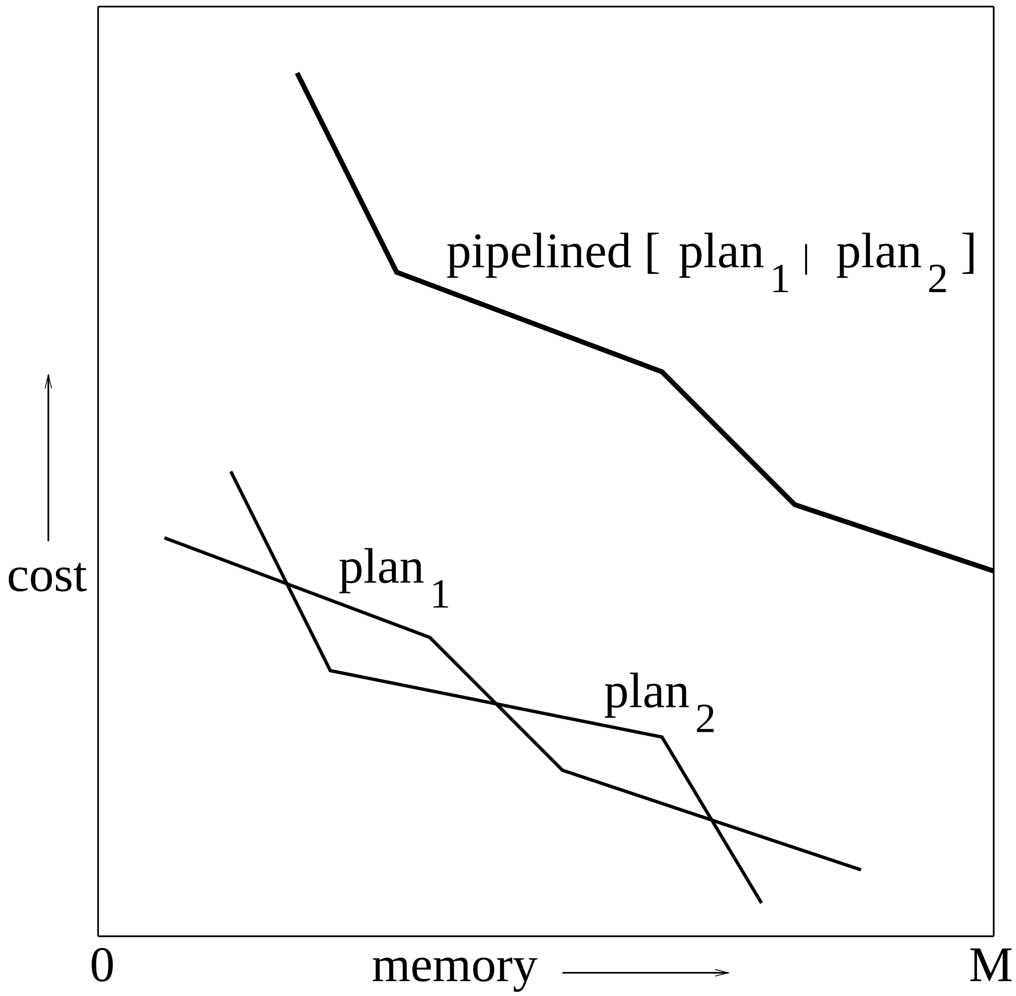
Dividing Memory between Operators

- Consider a pipeline tree P composed of two operators o_1 and o_2 with cost functions $c_1(m)$ and $c_2(m)$ resp.
- We can divide the total available memory M between o_1 and o_2 as m_1 units to o_1 and $m_2 = M - m_1$ units to o_2.
- The optimal division, say $m_{1_{opt}}$ units to o_1 and $m_{2_{opt}} = M - m_{1_{opt}}$ units to o_2 is given by:

$$\forall i, 1 \leq i \leq M, \forall j, 1 \leq j \leq M :$$
$$i + j = M \Rightarrow c_1(i) + c_2(j) \geq c_1(m_{1_{opt}}) + c_2(m_{2_{opt}})$$

Optimal Division of Available Memory

function $OptMerge$



Input: Cost functions for plan1 and plan2

Output: Cost function for the combined pipeline with division of memory amongst two subplans

Extended Plan and Cost Function

Extended Plan

- The optimal *Plan* for a query will contain a list of pairs: $\{[m_1, m_2], optPlan\}$
- Where $[m_1, m_2]$ is a range of memory sizes and $optPlan$ is the optimal plan for that range.

Extended Cost Function

Define cost as a function of memory allocated instead of a single cost value.

- General Form of Cost Function
- Piecewise Linear Cost Function

Extensions to Query Optimizer

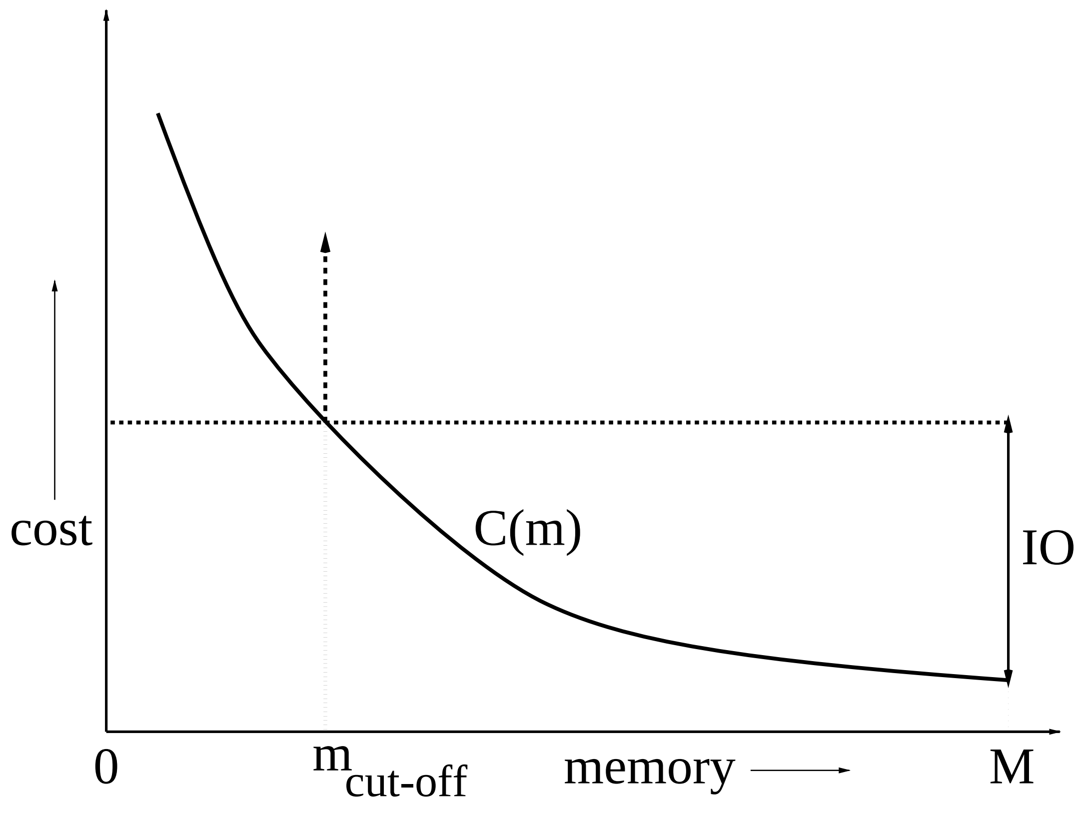
- Comparing alternative operators or plans
- Evaluating cost of a plan P composed of a root operator O and a subtree $P' = P \setminus O$
  - Edge between O and P' is blocked
  - Edge between O and P' is pipelined

Breaking Pipelined Edges

Consider a sub-plan P feeding an operator O through a pipeline edge E.

Options of running the sub-plan P:

- The edge E works as a pipeline edge
- The edge E works as a blocking edge



Current Status

We are in process of extending the Volcano Query Optimizer to make it Memory Cognizant.

Future Work

- Issues with Response Time in Memory Cognizant Query Optimization.
- Parametric Query Optimization.
- Memory Cognizant Query Optimization for e-ADTs.