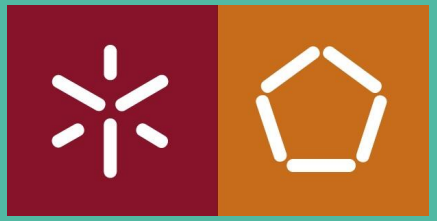


Algoritmo Partitioned-Cube

Bruno Ribeiro
Gil Gonçalves
Luis Pedro
José Monteiro

Universidade do Minho
Escola de Engenharia
Business Intelligence
Processamento Analítico de Dados

2017

Conteúdo

- Introdução
- Tipos de Métodos
- Métodos de Computação
- Algoritmo Partitioned-Cube
- Algoritmo Memory-Cube
- Conclusão
- Recursos Bibliográficos

Introdução

- As atividades de tomada de decisão que os agentes desenvolvem são na maior parte das vezes críticas, necessitando de ser suportadas por grandes volumes de dados
- Sendo as queries sobre os sistemas de dados operacionais um dos mecanismos utilizados pelos agentes de decisão para as suas atividades, estas têm de ter um tempo de resposta curto, algo que nem sempre é possível

Introdução

- A solução é a construção de cubos OLAP, na qual existem um conjunto de métodos para uma construção eficiente dos mesmos, tendo como objetivo tornar mais melhorar todo o processo de tomada de decisão

- Um cubo OLAP é uma estrutura de dados multidimensional construída a partir das dimensões de análise definidas, na qual cada uma das suas células contém um conjunto de valores (medidas), referenciadas por um conjunto de coordenadas que apontam para instâncias, dimensões ou níveis

Tipos de Métodos

- Métodos de Computação: abordam o problema de como o cubo OLAP deve ser computado
- Métodos de Seleção: abordam o problema de quais partes do cubo OLAP devem ser armazenadas
- Métodos Híbridos: métodos que integram ambos os problemas
  - computam o cubo e condensam o resultado da computação ao mesmo tempo
  - computação mais rápida é combinada com um armazenamento mais eficiente, conduzindo a melhores resultados

Métodos de Computação

- Percorrem os dados, aplicando as funções de agregação adequadas em todos os subconjuntos da *lattice*, gerando assim o cubo
- O objetivo principal prende-se a agregar todos os tuplos com os mesmo valores de atributos de agrupamento, colocando-os em posições de memória adjacentes
- As duas principais alternativas usadas são o *sorting* e o *hashing*

Métodos de Computação

- Operações de *sorting* e de *hashing* usadas para as agregações são mais eficientes quando os dados a processar cabem em memória
- Assim, a maior parte deste algoritmos partem os dados em fragmentos disjuntos que cabem em memória, chamados partições
- No entanto, estes tuplos não são colocados aleatoriamente em partições, sendo que dados que se podem agregar devem pertencer à mesma partição

Algoritmo Partitioned-Cube

- Baseado na estratégia *divide-and-conquer*
- Recursivamente “parte” as tabelas em fragmentos que cabem em memória, aplicando as operações a cada um desses fragmentos independentemente
- Este mecanismo de partição começa inicialmente pela tabela de factos, e executa recursivamente até que todas as partições caibam em memória
- Para cada partição, é aplicado o algoritmo Memory-Cube, que computa todos os subcubos em memória

Algoritmo Partitioned-Cube

- Inicialmente, parte a tabela de factos R em n fragmentos baseado no valor de um atributo B, e recursivamente chama-se n vezes passando por input cada uma das partições à vez
- A união de n resultados dá origem ao subcubo F, que consiste em todos os nodos que contêm B nos seus atributos de agrupamento
- De seguida, faz uma nova chamada recursiva dando como input o subcubo F, fixando o valor de B como ALL. Assim, é computado um subcubo D que consiste em todos os nodos que não contêm B nos seus atributos de agrupamento

Algoritmo Partitioned-Cube

Input:

- Conjunto de tuplos R
- atributos CUBE BY $\{B_1,...,B_m\}$
- atributo A a agregar
- função de agregação G(.)

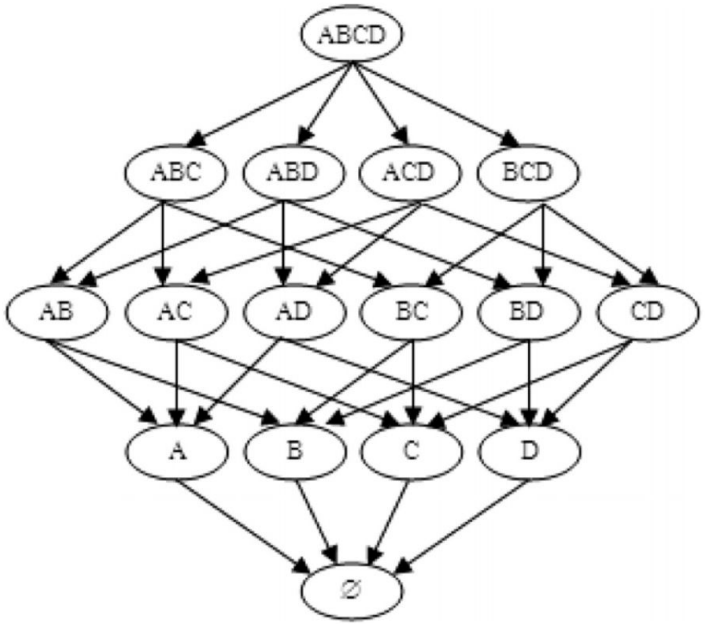
Output:

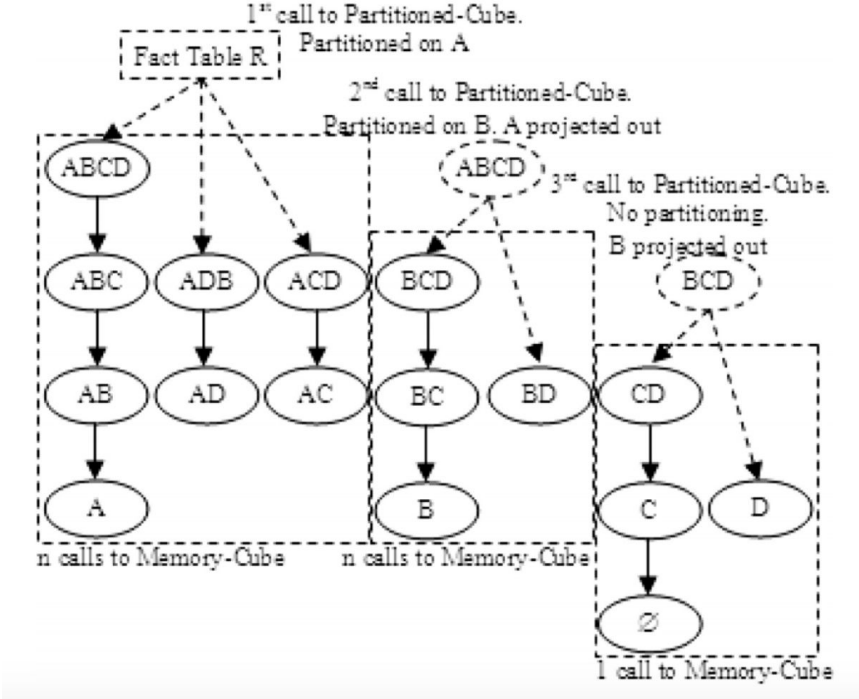
- F contém a granularidade mais fina dos tuplos do cubo
- D contém os restantes tuplos

Algorithm 1 Algorithm Partitioned-Cube

```
1:  procedure PARTITIONED-CUBE(R, {B_1, ..., B_m}, A, G)
2:      if (R fits in memory) then
3:          return Memory-Cube(R, {B_1, ..., B_m}, A, G);
4:      else
5:          choose an attribute B_j among {B_1, ..., B_m};
6:          scan R and partition on B_j into sets of tuples R_1, ..., R_n;
7:          /* n ≤ card(B_j) and n ≤ number of buffers in memory */
8:          for i = 1 ... n do
9:              let (F_i, D_i) = Partitioned-Cube(R_i, {B_1, ..., B_m}, A, G);
10:         end for
11:         let F = the union of the F_i's;
12:         let (F', D') = Partitioned-Cube(F, {B_1, ..., B_{j-1}, B_{j+1}, ...B_m}, A, G);
13:         let D = the union of F', D' and the D_i's;
14:         return (F,D);
15:     end if
16: end procedure
```

Algoritmo Partitioned-Cube





Algoritmo Memory-Cube

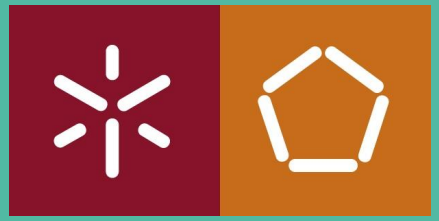
- Computa as partições fornecidas pelo algoritmo Partitioned-Cube, com a particularidade de a computação ser feita em memória
- Utiliza a estratégia de *sorting*, sendo que o número de operações de ordenação é igual ao número de caminhos que são formados percorrendo a *lattice*
- Este algoritmo tira partido das dependências entre operações de ordenação consecutivas, partilhando um esforço computacional considerável

Conclusão

- O objetivo do algoritmo Partitioned-Cube é combinar as vantagens dos algoritmos já existentes com uma melhor forma de partir os dados, resultando numa redução do I/O e numa computação mais rápida do cubo
- O custo de I/O é tipicamente proporcional ao número de dimensões, e não exponencial ou quadrático como nos algoritmos PipeSort e Overlap, respetivamente
- A computação do cubo feita em memória pelo algoritmo Memory -Cube contribui para a elevada performance do algoritmo

Recursos Bibliográficos

- Morfonios, K. et al., 2007, ROLAP implementations of the data cube. *ACM Computing Surveys*, 39(4), p.12-es.
- Ross, K. and Srivastava, D., 1997, Fast Computation of Sparse Datacubes. *Morgan Kaufmann Publishers Inc*, p.116-125



Algoritmo Partitioned-Cube

Bruno Ribeiro
Gil Gonçalves
Luis Pedro
José Monteiro

Universidade do Minho
Escola de Engenharia
Business Intelligence
Processamento Analítico de Dados

2017