Escalonamento de transações

Eduardo Mathias de Souza
GRR20190428

O trabalho consiste na implementação em C de dois algoritmos de detecção de conflitos de escalonamento de transações concorrentes. A entrada do algoritmo, lida do stdin, é formada por uma sequência de linhas com 4 campos: tempo de chegada, identificador da transação, operação e atributo que será lido/escrito.

Cada linha representa uma transação nova, podem ter inúmeras distintas transações (transações com IDs distintos) ativas, e as operações que cada transação pode realizar é Read (Ler), Write (Escrever) ou Commit(Commitar/Finalizar).

No algoritmo criado, apresento um array de structs transação, onde guardo cada transação nova com seus respectivos campos de ID, operação, atributo. Ainda escolhi optar por colocar um campo para indicar se a ação é um commit ou não.

Essa struct é apresentada então assim:

```
struct Transacao{
	int t; //Representa ID da  transação
	char acao; //Representa a ação da transação nova(R, W, C)
	char variavel; //Representa o atributo/variável que a transação realizará a ação
	int Commit; //Se a ação da transação é um commit
};
```

Quando uma transação realiza o commit, verifico se as outras transações ativas também realizaram, se sim, passa-se a ser feita a detecção de conflitos de escalonamento nessa “linha” de transações concorrentes.

Para realizar a detecção de conflitos temos dois algoritmos:

Algoritmo de teste de seriabilidade quanto ao conflito:

- Crie um nó para cada transação com id distinta analisada
- Cria-se Aresta T_i -> T_j para cada r(x) em T_j depois de w(x) em T_i
- Cria-se Aresta T_i -> T_j para cada w(x) em T_j depois de r(x) em T_i
- Cria-se Aresta T_i -> T_j para cada w(x) em T_j depois de w(x) em T_i
- S é serial se não existe ciclo no grafo

Para implementar esse algoritmo, utilizei estruturas de Grafo com nodos e Lista de adjacência. Cada transação com id distinta era um nodo, e, possuía uma lista de adjacência.
A lista de adjacência representa os nodos que tal nodo é ligado e servirá posteriormente para verificar se há ciclo no grafo.
Por exemplo, se T1 possui uma aresta T1->T2, o nodo T1 terá em sua lista de adjacência T2

Após montado o Grafo, utilizo o algoritmo detecta_ciclo e o algoritmo testa_ciclo (chamado dentro de detecta_ciclo para cada vértice do Grafo e chamado em si recursivamente[para andar dentro das arestas das listas de adjacência]) para rodar por todo o grafo com dois arrays auxiliar (visitou e path)

e analisar se tem ciclo. Caso tenha ciclo no grafo, retorna 1 e dizemos que o escalonamento não é serial, caso contrário, retorna 0 e dizemos que o escalonamento é serial.

Algoritmo de visão equivalente

Dois agendamentos S e S' são ditos visão equivalentes, se atenderem às seguintes condições:

- O mesmo conjunto de transações e operações estão presentes tanto em S e S'
- Para cada atributo x lido por T_i, se o mesmo atributo for escrito por T_j em S, essa mesma sequência deve permanecer em S'
- Se o atributo y for escrito por T_k, e essa for a última escrita desse atributo em S, então a última escrita do mesmo em S' deve ser feita por T_k tbm

Para implementar esse algoritmo, criei um array que armazena todos os atributos do escalonamento analisado. Para cada atributo, vejo para qual T era a última escrita do mesmo e em qual instante de tempo.

Se existir leitura do mesmo atributo no T que foi o último a escrever ele, nos instantes de tempo anteriores desta última escrita, com escrita do mesmo atributo em T distinto.
Ou seja, se existe Ti r(x) -> Tj w(x) -> Ti w(x), não existirá um agendamento que será equivalente, diante de todas as permutações existentes de S.
Além disso, criei no meu algoritmo de visão equivalente um Grafo com listas de adjacência chamado GrafoVisão para controlar qual Transação deve vir antes de qual. Isso ajuda no controle de ver se alguma transação é equivalente quando temos mais de um atributo.
Por exemplo: se T1 for o último a escrever em x, porém acima de R(X), T1 realiza um R(y), e, T2 escreve em x e le em x porém é o último a escrever em y. Pela regra 3 acima, T1 deve vir depois de T2, porém pela regra 2 T2 deve vir depois de T1. Logo temos um conflito/ciclo e não é equivalente o escalonamento
Logo, nesse GrafoVisão, cada nodo representa a transação e na lista de adjacência quem deve vir antes dela. Se GrafoVisão tem ciclo (testado pelo mesmo algoritmo que uso em Monta Seriabilidade) não é equivalente o escalonamento.

Logo, caso ocorra o descrito acima, retorna 0 e dizemos que o escalonamento não tem visão equivalente. Caso contrário, dizemos que é.

A saída do algoritmo é feita pela saída padrão (stdout). É composto por uma sequência de linhas, na qual uma linha representa um escalonamento e tem 4 campos separados por um espaço: O primeiro campo é o identificador do escalonamento, o segundo campo é a lista de transações, o terceiro é o resultado do algoritmo da garantia da seriabilidade, na qual “SS” é serial e “NS” é não serial, e, o quarto campo é o resultado do algoritmo de teste de equivalência de visão, na qual “SV” é equivalente e NV é não equivalente