Universidade Estadual de Santa Cruz

Indíces Avançados

Eduardo Reis Nobre

11 de junho de 2018

Sumário

1 Introdução

O uso de de índices trazem a capacidade de organizar uma tabela de forma a trazer uma maior velocidade para consultas feitas. Outras técnicas podem ser aplicadas para atender outras necessidades. Índices por padrão, não afetam a organização dos dados em si, eles são apenas uma forma de se chegar ao dado de forma mais eficaz. Ao se utilizar clusters os dados em si são reorganizados. A localização física dos dados é um fator a ser considerado quando lidamos com bancos de dados contendo grandes quantidade de de dados.

Ao expandirmos nossos conhecimentos sobre índices, vemos que índices podem ser clusterizados ou não, sendo por padrão, não clusterizados. A criação de um cluster é feita sob um índice já existente. Outras técnicas podem ser utilizadas para otimizar como os dados se encontram organizados no disco. Comandos como reindex e vacuum visam lidar com alterações realizadas em uma tabela reorganizando reconstruindo índices, criando novas conexões considerando novos dados assim como removendo espaços em branco entre dados, causados por DELETES e operações similares.

Esse relatório trata do uso dessas técnicas e seu impacto na velocidade das consultas realizadas sob uma base de dados contendo 2839 registros.

2 Materiais de Métodos

2.1 Ambiente

Os [**?**] testes foram realizados utilizando um notebook com a seguinte configuração:

Processador: Quad-Core Intel® Core™ i5-8250U CPU @ 1.60GHz
Memória Ram: 8Gb @ 2400MHz
HD: Western Digital 1TB, Leitura 176 MB/s, Escrita 167MB/s
SO: Elementary OS 0.4 Loki

Os testes foram realizados utilizando-se o sgbd postgres versão 9.5.

2.2 Metodologia

3 Metodologia

Dois parâmetros serão utilizados para avaliar a saúde e performance de nosso banco, correlação e tempo médio levado para se realizar uma operação.
O primeiro deles, pode ser encontrado na tabela pg_stats do banco de dados, e se trata da correlação estatística entre a ordem física de uma coluna e a ordenação lógica dos valores de uma coluna. Esse valor varia de +1 a -1. Quando esse valor se encontra próximo de +1 ou -1, considera-se que uma busca realizada nesta coluna é mais rápida do que se o valor estivesse próximo de zero por conta da redução dos acessos aleatórios ao disco. O segundo é a média aritmética dos tempos dos testes realizados.
Podemos acessar a correlação através do script abaixo

```
SELECT correlation, attname FROM pg_stats WHERE tablename =
    't_pescador';
```

Temos como resultados iniciais os valores abaixo:

correlation	*attname*
0.287947	tp_id
0.576103	tp_sexo
0.0296074	tp_nome

Realizando a consulta abaixo 10 vezes, sem o uso de indíces para a coluna **tp_id** temos o resultado de tempo médio,de **0.6087**ms

```
SELECT * FROM t_pescador where tp_id = 1831;
```

3.1 Clusters

Ao criar um cluster, os dados são reorganizados de forma ordenada no disco, após a criação de um cluster, caso um novo dado seja inserido, ele será inserido no fim da tabela. Por conta dessa característica, o uso de clusters não é indicado para para tabelas que sofrem uma grande quantidade de alterações em seus dados, sendo indicado principalmente que recebem uma grande quantidade de pesquisas, mas que sem mantém inalteradas por um longo período de tempo, como lista de Estados, Países, Departamentos e afins.

Através do uso de um cluster, ordenamos perfeitamente de acordo com qual índice foi utilizado para a criação do cluster. Como a natureza do cluster

está ligada à organização física dos dados em disco, apenas um cluster pode ser mantido ao mesmo tempo para uma dada tabela.

O script abaixo é encarregado da pela criação do cluster.

```
CREATE INDEX tp_id_index ON t_pescador USING btree(tp_id);
CLUSTER t_pescador USING tp_id_index;
```

Ao verificarmos a correlação novamente temos os seguintes resultados:

correlation	*attname*
1.0	tp_id
0.576304	tp_sexo
-0.00931414	tp_nome

Como já esperado, ao utilizar um cluster, a coluna tp_id foi ordenada em disco, trazendo nosso tempo médio para **0.5695**ms, uma pequena melhoria, mas que em bancos de dados de tamanho significativo, traz uma melhoria de performance.
Como citado na introdução desse relatório, o cluster perde performance quando operações de inserção e remoção são realizadas na tabela, fazendo com que o indíce que fora de ordm. Essas operações além de afetar o cluster, também causam que nossos indexes sofram sofram causando uma perca de performance.

3.2 Reindex

Para que que possamos ver os efeitos causados por inserções o script abaixo foi aplicado.

```
-- Cria uma tabela baseada nem 25% da tabela t_pescador
CREATE TABLE t_pescador_back AS SELECT * FROM t_pescador ORDER BY
    random() limit (SELECT (count(*)*0.25)::INTEGER FROM t_pescador);

-- Cria uma cria uma sequncia para ser utilziada como autoincrement
    do tp_id e evitar que na insero, o banco coloque os dados no
    mesmo lugar onde eles se encontravam.
DROP SEQUENCE t_pescador_seq_id cascade;
CREATE SEQUENCE t_pescador_seq_id;
ALTER TABLE t_pescador
ALTER COLUMN tp_id SET DEFAULT nextval('t_pescador_seq_id');
SELECT setval('t_pescador_seq_id', (SELECT max(tp_id) FROM
    t_pescador));
```

```
-- Remove registros da tabela t_pescador baseado nos registros
    inseridos na tabela t_pescador_back;
DELETE FROM t_pescador WHERE tp_id IN (SELECT tp_id from
    t_pescador_back);

-- Insere dados da tabela t_pescador_back omitindo os os ids
INSERT INTO t_pescador (tp_nome, tp_sexo) (SELECT tp_nome, tp_sexo
    FROM t_pescador_back)
```

O script acima não afeta a correlação da coluna tp_id, pois os os id's foram criados pela **SEQUENCE** criada acima, porém afetou ligeramente os tempos. Isso ocorre, pois o index foi alterado e não se encontra otimizado no momento. Ao fazermos nossa consulta novamente temos um tempo médio de **0.6209**ms.
Ao utilizarmos **reindex** o index é recriado completamente, criando caminhos otimizados para o novo estado do banco reduzindo nosso tempo méido para **0.5785**ms.

3.3 Vacuum

Nosso script acima, além de afetar os indices, também criou espaços em branco onde os registros foram removidos. Para resolver esse problema podemos usar o comando **VACUUM**. Esse comando remove os espaços em branco, desfragmentando o banco de dados e melhorando a performance reduzindo nossos tempos para **0.5368**ms.

4 Considerações finais e resultados

Para chegar a uma conclusão, as etapas acima foram definidas como experimentos sendo eles: *no_index*, consulta realizada sem indice algum; *index+cluster*, consultas realizadas com índice para tp_id + cluster; *disturb* consultas criadas após criação da perturbação no banco de dados; *reindex* consultas realizadas após reindexação; *vacuum* consultas realizadas após remoções de espaços em brancos.

A primeira coluna da tabela de resultados representa o experimento em questão, a segunda o tempo médio das consultas durante o experimento, a terceira e quarta coluna comparam a diferença entre experimentos, verificando a diferença de tempo entre o experimento em questão e o experimento anterior apresentando essa diferença em milissegundos e em percentagem respectivamente.

Experimento	*Tempo em ms*	*Speedup*	*%*
no_index	0.6087	-	-
index + cluster	0.5695	-0.0392	-6.44
disturb	0.6209	+0.0514	+9.03
reindex	0.5785	-0.0424	-6.83
vacuum	0.5368	-0.0417	-7.21

Um dos resultados dignos de nota foi como uma perturbação em 25% do banco de dados foi o suficiente para fazer com que a sua o banco tivesse uma performance inferior a um banco sem índices algum, reforçando a importância das rotinas de reindexação assim como destacou o impacto que espaços vazios entre dados podem causar em um banco já que operação de vacuum foi que causou maior impacto positivo no desempenho.

https://github.com/reisnobre/db_2_rel03: Link para repositório contendo tempos e scripts assim como resultados da primeira parte do relatório.