Análise:
"A Practical Scalable Distributed B-Tree"

Iyan Lucas Duarte Marques[1]

[1]Instituto de Ciências Exatas e Informática — Pontifícia Universidade Católica Minas Gerais (PUC-MG)

1. Introdução

O artigo escrito por Aguilera, Golab e Shah apresenta uma árvore B+ distribuída cuja proposta é escalabilidade, alta taxa de transferência e grande armazenamento. A estrutura é focada em servidores para alta eficiência, onde vários *nodes* são localizados em múltiplos servidores. A utilização da mesma varia de, igual dito no artigo, banco de dados e aplicações *online* à (jogos) *multiplayer online*.

2. *B+ Tree vs B+ Distributed Tree*

Apesar das aplicações *online* e a questão da distribuição de *nodes* entre vários servidores, o que me impressionou foi a diferença entre os métodos natívos da estrutura. Mais especificamente os métodos: *getRoot()* e *Alloc() Free()*[1] Como Aguilera diz nessa parte, a árvore pode sofrer modificações ao longo da sua utilização, como quebra de folhas, mudança de altura, entre outros. Por isso, diferentemente da árvore tradicional, a árvore distribuída guarda um objeto especial contendo a meta data da raiz e uma lista de todos os servidores que essa estrutura se encontra, garantindo uma grande eficiência.

Em relação à alocação de nodos da estrutura, visto que a mesma se encontra em vários servidores e recebendo operações em tempo real, há a necessidade de evitar múltiplas alocações ao mesmo tempo[2] e *memory leaks*[3]. Desta forma, em cada servidor tem um número de nodos inutilizados e um vetor indicando qual nodo está disponível. O qual cada cliente tem uma duplicata para garantir mais eficiência. Logo, o cliente seleciona o servidor através de uma função que retorna o mais ideal e seleciona um nodo que está livre. A desalocação é similar à alocação, somente a parte de seleção do nodo que ocorre inversamente. Isto se difere da árvore tradicional que não possui esse vetor e essa *pool* de nodos, se tornando assim mais vulnerável à falha e consequentemente mais suscetível a erros.

[1]Seção 6, (DETAILS OF DISTRIBUTED B-TREE ALGORITHIM), subseção 6.1 (Dictionary and enumeration operations), respectivamente: *Finding the root* e *Node allocation and deallocation*

[2]Quando a mesma operação de alocação é chamada mais de uma vez, gerando assim um conflito ou mais de uma alocação do mesmo objeto, gerando uma redundância prejudicial.

[3]Quando a operação de alocação é abortada em tempo de execução.