Redistribution

- number of nodes is unchanged during redistribution.

- effects are strictly local (at node and one sibling).

- no fixed prescription for redistribution. If n is order of B-tree, and a node, p, has ceil(n/2) descendants (which is the min number) and a sibling with n descendants, then we redistribute by moving anywhere from 1 to n - ceil(n/2) descendants from the sibling to the node, p.

- usual strategy is to redistribute evenly. Why?

- why not redistribute during insertion? What would be the effect?

- empirical results show that w/o redistribution during insertion, space efficiency (ratio of space used to store info to space to hold total B-tree) is approx. 67% (theoretically 69%), while with redistribution, efficiency can improve to 86%.

B*-Trees

- consider system that allows redistribution during insertion. Thus, when a given node needs to be split, we can guarantee that at least one sibling is already full. How?

- Take the two full nodes and split to three.

- each page therefore has at least ceil(2m-1)/3 descendants.

- the root has at least two descendants

- all leaves are on same level.

Virtual B-Trees

- keeps as much of B-tree in memory as possible instead of just root.
- create a page buffer to hold B-tree pages.
- what to do in case of page fault? What page do you replace?
- most general approach is LRU strategy, i.e., replace the one that has been unused for longest period.

- assumes temporal locality.

- rather than LRU, could replace based on page height, favouring retention of higher pages in the index.

- can use combination of the above approaches. How?