By: Nicholas Soucy

For: Dr Roy Turner

B-Tree Performance

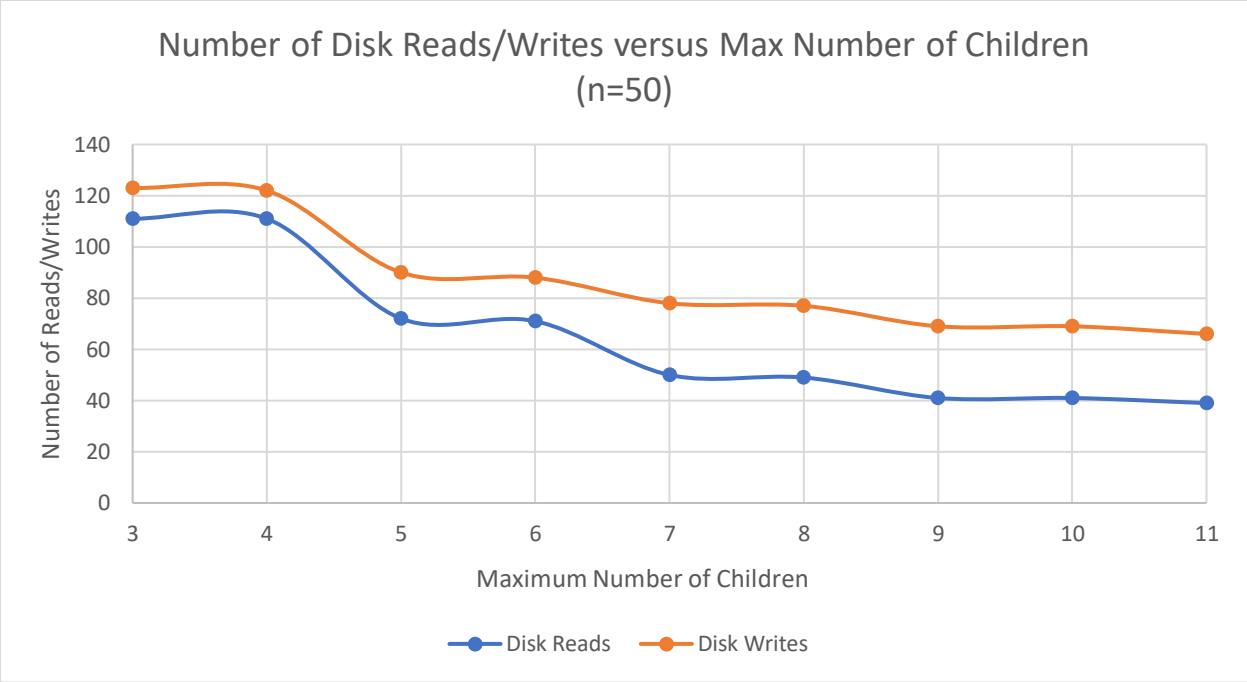
Disk Reads & Writes

Throughout the test, it was clear that the larger the maximum children per node was, the less disk reads and disk writes there were.

This makes sense for disk write, because splitting a node cost three disk writes, and you need to split a child when you insert into a full node. When you decrease the maximum number of children per node and keep the amount of data a constant (n), you need to split more frequently. Therefore, there will be more disk writes.

For disk read, there is a disk read in the insert into a nonfull node when that node is not a leaf node. This disk read accounts for most of the disk read throughout the program. As we increase the maximum number of children per node, there will be more and more leaf nodes because they can hold more data per node, therefore there will be less calls for disk read while inserting into nonfull nodes. This explains the tread of decreasing disk reads as the maximum children per node increases.

Below is a graph including the number of disk writes and disk reads as a function of maximum children per node. Each data point is an average of 50 runs with a random list of numbers each time, but with a constant size of n = 50 and a constant range of numbers 0 – 500.



Disk reads and writes are an expensive process that is order of magnitudes slower than memory reads and writes. If we had actual reads and writes instead of counters, we would have a much slower program on our hands, especially for large data sets.

Virtual Memory

The point of B-Trees is to solve the problem of storing large volume data that is too large to fit into memory all at once. To solve that issue, the B-Tree gets stored on many disk pages, and some of them might be in memory when others are on disk. We want to minimize the amount of times we have to go to disk because that will create a page fault. This process requires a lot of time, so we want to minimize the amount of page faults if possible.

For B-Trees with a small maximum children per node, like a binary tree or a 2-3 tree, page faults can be quite high. We can see this with the data outlined in the graph above. Theoretically, we can predict this because for a 2-3 tree with n data points, the number of comparisons required to insert a new piece of data will take approximately $\log_3 n$ comparisons. If n = 1,000,000, that is 13 comparisons, which is possibly 13 page faults! We want to minimize this as much as possible. Therefore, higher maximum children per node B-Trees, like 2-7 Trees, are used

more often in these circumstances, they would take that 13 comparisons down to $\log_7(1{,}000{,}000) = 8$ comparisons!

In conclusion, these Disk Reads & Writes are simulations of page faults in the virtual memory system. From the data of the simulation outlined in the graph above, we can see how larger maximum children per node B-Trees will decrease the height of the B-Tree and create potential for fewer page faults.