B-tree Page Size When Caching is Considered

David Lomet

Microsoft Research

Redmond, WA 98052

lomet@microsoft.com

1. Introduction

The recent article by Gray and Graefe in the Sigmod Record [1] included a study of B-tree page size and the trade-off between page size and the cost of accessing B-tree data. Its goal was to find the page size that would result in the lowest access cost per record. This insightful analysis showed how increasing page size permitted the B-tree to be traversed faster while increasing the amount of data that needed to be read to perform the traversal. Their analysis captures the trade-off between the cost (in time) of each access and how many accesses are needed to traverse the tree.

What the analysis in [1] does not capture is the impact on B-tree cost-performance that results from caching parts of the index tree in main memory. Substantial parts of a B-tree index (above the leaves) will often be memory resident. It does not matter what the size of those pages is. What does matter is how much memory is required to hold that part of the B-tree, and what the access cost is for the residual accesses.

To account for the effect of main memory caching of parts of the B-tree index, we proceed in two steps. Section 2 determines how much of the B-tree index can cost effectively be cached in main memory. Section 3 then determines how the size of the B-tree pages affects the cost-performance of the residual accesses. Section 4 applies the analysis to the bounded disorder access method. A short discussion section concludes the paper.

Notation:

We introduce the following notation for this.

D	disk size in bytes
P	page size in bytes
E	index entry size in bytes
F	index node fanout
S	disk latency in seconds
B	disk bandwidth (transfer rate) in bytes/sec
C_M	\$/Mbyte of main memory
C_D	\$/Mbyte of disk memory
A	accesses/second per disk
u	storage utilization for B-tree

One has to purchase a disk in order to acquire A accesses/second. Thus, we determine the cost per A access/sec as C_D*D.

2. How much of the Index to Cache

In this section, we want to show when it is cost effective to provide some level of caching so as to realize retrieval savings. Essentially, we want to understand when the cost of memory for improving our access rate to records of the B-tree is less than the cost of purchasing additional access arms. Of particular interest is how many levels of the index we can reasonably anticipate caching.

Now we narrow our focus to determining when it is economically justified to keep the bottom level of the index (just above the data) in main memory. Our assumption continues to be that the access pattern to the data is uniform. This analysis easily extends to higher levels of the index, where it is readily seen that keeping these levels in main memory always pays off.

We assume that the disk is processing our requests for records at its maximum access rate *A*. This rate involves two disk accesses per record request, one for the page at the bottom level of the index and one for the page containing the data. The question to be asked is "should we buy another disk or should we add memory capacity so as to put the bottom level of the index into our cache?" We want to avoid being heavily subject to granularity issues here. So we imagine that we can buy fractional accesses/second capacity via disk purchasing as well as via additional index caching.

We also assume here that our data fills the disk. The index size (above the leaves) is less than 1% of the size of the leaves). As the disk size increases then, this will increase the memory size needed for caching, e.g., the bottom level of the index. It will also increase the access arm cost compared with the cost/byte of disk storage. Both these are assumed linear, and hence the caching payoff is driven by the relative costs of disk and main memory.

We understand that the above assumptions (approximations) are crude. However, this should not impact the main lessons to be learned from our analysis.

Primary B-trees

We deal here with B-trees acting as primary (clustered) indexes, and assume the disk is filled with the indexed data. We assume further that caching all levels of the B-tree but the bottom two (i.e. the last index level plus the data level) is cost effective. We then show that the performance gain is sufficiently large from caching the last index level to make caching that level cost-effective as well.

The number of data pages (we ignore index pages, which are less than 1% of the total) is D/P. This is also the number of index terms (<key,pointer> pairs) in the level of the index just above the leaves. Each index term takes up E/u bytes. We multiply the number of index terms times the size of each term to determine the amount of main memory taken up by the index.

$$\text{Index memory} = (D/P)(E/u)$$

$$\text{Caching Cost} = C_M\ (\text{Index memory})$$

Without caching the level of the index just above the leaves, the cost of accessing a record is two disk accesses, one for the level above the leaves, the second for accessing the leaf data node. Hence, caching the level of the index above the leaves doubles the number of record accesses that the combined system can support. This is equivalent to adding another disk at $C_D D$ dollars.

We want caching cost to be less than disk arm cost if we are to justify caching the index in main memory. Thus, for caching of the lowest level of the index to be effective, it needs to cost less than the cost of a disk arm:

$$C_M\ ((D/P)(E/u)) < C_D D$$

or

$$C_M/C_D < u(P/E)$$

or

$$P > (C_M/C_D)(E/u)$$

So now let's look at some costs based on present day prices. [Costs change continually, so these are surely already out-of-date.]

$$C_M = \$3/MB \quad (\$50 \text{ for } 16MB)$$
$$C_D = \$.06/MB\ (\$250 \text{ for } 4GB)$$

At those costs, $C_M/C_D = 50$. If $E = 20$ bytes and u = .667, then E/u is *30* bytes, and P must be larger than *1500* bytes. Obviously, this is satisfied by all B-trees of any interest. (Note here that index node size is actually irrelevant. It plays no role in determining what is the size of the bottom level of the index. Only data node size impacts the outcome.)

Another way to look at this is that we are spending 30 bytes of main memory per page of disk that we are accessing to cut the number of disk accesses/page in half.

Secondary B-trees

When dealing with B-trees used as secondary (non-clustered) indexes, the argument for caching all but the leaf level nodes in main memory continues to hold. However, we are two disk accesses from the primary data, as the leaves contain not the records but <key, pointer> pairs with the pointer indicating where the record is. So the question here is, is it cost effective to cache the leaves of a non-clustered B-tree (so that essentially the entire tree is cached) in order to be a single disk access from the primary records. The answer to that question will usually be "no".

There is a <key, pointer> pair for every record. Thus, the ratio of data size to index size is R/E where R is the size of the records being indexed. This is much lower than the case for primary B-trees. Typically, R is of the order of a couple of hundred bytes, say *240* bytes for easy arithmetic. With an E of 20 bytes, $R/E = 12$, which is less than C_M/C_D. The record size R must be of size 1000 bytes and a factor of 50 larger than the space consumed by an index term, for us to profitably cache this bottom B-tree level. Hence, we conclude that it is usually not worthwhile to store the bottom level of a non-clustered index. Note also that while we can control the fanout of a primary B-tree via changing the page size, that the fanout of a non-clustered index is not under our control.

3. The Residual Accesses

Analysis

In the analysis above, it was shown that it is **ALWAYS** profitable to keep all but the leaves in main memory, so long as page size is at least *1500* bytes. However, what was not shown was what page size minimized the total cost (memory plus disk) for accessing a record. Given the above results, we assume here that all of the non-leaf levels of the B-tree are in main memory. We then calculate the cost of dedicating main memory for this part of the tree. We balance this against the cost per accesses/sec that we can achieve with a given page size. Essentially, the trade-off is that the larger the page, the lower our supportable access rate, and hence the higher the cost of our accesses/sec. However, the larger the page, the smaller the main memory needed to hold the upper part of the index. So here the trade-off is between the cost of memory for caching and the impact of page size on disk access rate.

As before, $(E/u)(D/P)$ is the size of our main memory buffer, and memory cost is $C_M(E/u)(D/P)$. This memory must be dedicated to serving the B-tree. It saves accesses at whatever the rate is that the disk can sustain. We have set things up so that it takes one disk access to get to a record. Thus, the record access rate is the disk access rate. It is

$$\text{Access rate} = 1/(\text{access time}) = 1/(S+P/B)$$

Note that this access rate is dependent upon page size.
We need now to compute the cost of accesses/second. Both memory and disk contribute to providing this access rate.
Our cost for accessing the disk via a B-tree whose index levels are all in main memory is the main memory cost of caching the index plus the cost of the disk. This is

$$Cost = C_M (E/u)(D/P) + C_D D$$

$$\begin{aligned} &Cost/(accesses/sec) = \\ &Cost/(1/(S+P/B)) = \\ &\qquad C_M (E/u)(D/P) S + \\ &\qquad \{C_M (E/u) D/B + C_D DS\} + \\ &\qquad (C_D D/B) P \end{aligned}$$

This is of the form $k_1/P + k_2 + k_3{*}P$

To find the minimum cost as a function of P, we can differentiate with respect to P and set the derivative to zero. This yields

$$- k_1/P^2 + k_3 = 0 \quad or\ P = (k_1/k_3)^{1/2}$$

Substituting original quantities back in yields

$$P = ((C_M/C_D)(E/u)\ SB)^{1/2}$$

Values for cost and performance are constantly changing. However, the ratio of main memory to disk cost changes more slowly than do the individual values. The entry size E is little affected by technology changes. It may have to be a bit larger as disks become larger, but this too changes very slowly. Thus, the greatest sensitivity on the optimal page size comes from the disk transfer rate. Below we provide some representative values.

$$\begin{aligned} &C_M = \$3/MB \quad (\$50 \text{ for } 16MB) \\ &C_D = \$.06/MB\ (\$250 \text{ for } 4GB) \\ &B = 10^7\ MB/sec \\ &S = 10\ ms = 10^{-2}\ sec \\ &E = 20 \text{ bytes } (E/u = 30 \text{ bytes at } u \\ &\qquad\qquad = .667) \end{aligned}$$

So

$$P = (50{*}30{*}10^{-2}{*}10^{7})^{1/2}\ bytes$$

$= (1.5*10^8)^{1/2}$ *bytes*
$= 1.25*10^4 = 12.5K$ *bytes*

The variability in this optimal value of P depends primarily on $S*B$, which is currently 10^5. This does change with technology because B, the transfer rate increases faster than S, the seek time, decreases. It was not so long ago that disks had seek times of *20* milliseconds and transfer rates *of* $.5*10^6$ bytes/sec, for a product $S*B$ of 10^4. This yields a decrease in page size by a factor of *three*. This means that in that era, 4KB pages were the optimal. (This depends on the assumption that the entire upper levels of the index were in main memory.) Indeed, 4KB pages were quite common then.

It is surely worth noting that the "answer" here for page size, i.e. *12.5KB*, is consistent with the result reported in [1]. This is somewhat fortuitous as at least some of the elements going into the analysis (e.g., cost and size of main memory buffers) are different. It does mean, however, that it is possible to choose a page size that is well tuned for both the caching and non-caching cases.

Calculating Costs

To calculate costs, we compute the values of our cost coefficients. We further assume here that we are dealing with a *4GB* disk, i.e., one with *4000 MB* capacity.

$$k_1 = C_M (E/u)DS = \$3600 \text{ bytes-secs}$$

$$k_2 = C_M (E/u)D/B + C_D DS = \$2.44 \text{secs}$$

$$k_3 = C_D D/B = \$ .24*10^{-4} \text{ sec/byte}$$

Thus,

$$Cost/(Accesses/s)\{in \ \$\text{-}secs\}= 3600/P + 2.44 + .24*10^{-4} P$$

For *12KB* pages, this is *.29 + 2.44 + .29 = $3.02*

Note that the variable cache cost and variable disk cost are "equal".

Changes from optimal page size skew these costs, smaller page size increasing cache costs, larger page size increasing disk costs. Compare this with the following:

- *4KB page, .88 + 2.44 + .098 = $3.42*
- *8KB page, .46 + 2.44 + .20 = $3.10*
- *16KB page, .23 + 2.44 + .39 = $3.06*
- *32KB page, .11 + 2.44 + .78 = $3.33*
- *64KB page, .06 + 2.44 + 1.56 = $4.06*

Thus one can see that the page size range from 8KB to 16KB bytes all have costs very near the minimum, and surely well within the uncertainty associated with this kind of analysis. However, it is pretty clear that outside of that range, that one is paying more per access/sec than is necessary or desirable.

4. Bounded Disorder Analysis

BD files[2] are B-trees in which the leaves of the tree are multi-block nodes. Only one block of the multi-block node needs to be read on a random access, and that is selected via hashing. Thus, each BD file leaf node is a small hash file. The effect of this that to determine the number of index terms (formerly *D/P*), one now has (*D/bP*) while the disk transfer size remains P where *b* is the number of buckets (pages) per leaf node.

Cost/(accesses/sec) then becomes:

$$C_M (E/u)(D/bP)S+ \{C_M (E/u)D/bB + C_D DS\} + C_D DP/B$$

This makes $k_1 = C_M (E/bu)DS$.

The optimal value for page size P becomes

$$P = ((C_M/C_D)(E/u)/(bSB))^{1/2}$$
$$= (1/b^{1/2})*(B\text{-tree opt. page size})$$

where *12.5KB* is the B-tree optimal page size.

With $b = 64$,

$$P = 12.5KB/8 = 1600B$$

The constant term is now

$$k_2 = \{C_M (E/u)D/bB + C_D DS\} = \$2.40$$

And cost becomes

$$Cost = 3600/(bP) + \$2.40 + .24*10^{-4} *P$$
$$= .035 + \$2.40 + .038 = \$2.47$$

This cost is dominated by the cost of the disk. One could set the caching cost to zero and change the cost by only three cents per accesses per second. So further improvements in caching effectiveness will not change the access rate to data until a substantial part of the data itself can be cached in main memory, not just the index.

The situation will change only with substantial reductions in seek latency. Indeed, the cost per access/sec is dominated by the term $C_D DS$. This is the cost of disk seek latency.

Bounded disorder indexing continues to have a pay-off, though with current technology, its advantage is only very marginally in its access rate, which will be only slightly higher because it can tolerate a smaller page size. What started out as an access method intended to save disk accesses has become an access method to reduce caching costs. (It saves cpu processing time as well by reducing the number of nodes that need to be searched on the path from root to leaf, a cost not included in our analysis.)

5. Discussion

The above analysis for B-trees suggests that either *8KB* or *16KB* pages should be fine. Further, the trend is for gradually increasing page size because $S*B$ looks like it will continue to increase gradually over time. Only the chance that main memory cost might become closer to disk memory costs would alter this course, permitting a larger cache and reducing disk transfer costs. But this by itself will not dramatically shrink the cost of disk accesses/sec. Only a dramatic improvement in disk seek time will accomplish that.

In closing, let me give some caveats with respect to this analysis. We have assumed the following.

- The disk is filled with B-tree data. This is not a large difficulty.
- The disk access arm cost is a function of disk size. This is not true and it matters. Smaller disks have higher cost per byte, and hence result in a lower C_M/C_D ratios. This pushes P toward smaller sizes. The reverse is true as disk size increases.
- At start up, larger pages (in the size range indicated by [1]) will have an advantage since the B-tree must be traversed when the cache is cold.
- I/O bandwidth is not the limiting resource. If it is, then smaller pages are preferred since they take less time on the bus.

One should regard the preceding analysis in the vein suggested by the title in [1]. It has produced a rule of thumb. This is useful for guiding your intuition and for making system trade-offs. But this analysis does not capture fine differences in performance. It is intended to be suggestive, not definitive.

Acknowledgments

Paul Larson and Goetz Graefe made helpful comments during the preparation of this technical note.

References

[1] Gray, J. and Graefe, G. The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. *ACM Sigmod Record* 26,4 (Dec. 1997) 63-68.

[2] Litwin, W. and Lomet, D. The Bounded Disorder Access Method. *Intl. Conf. on Data Engineering* , Los Angeles (1985) 38-48.