The Effect of Buffering on the Performance of R-Trees *

Scott T. Leutenegger Mario A. Lopez
Mathematics and Computer Science Department
University of Denver Denver, CO 80208-0189
{*leut,mlopez*}*@cs.du.edu*

Abstract

Past R-tree studies have focused on the number of nodes visited as a metric of query performance. Since database systems usually include a buffering mechanism we propose that the number of disk accesses is a more realistic measure of performance. We develop a buffer model to analyze the number of disk accesses required for spatial queries using R-trees. The model can be used to evaluate the quality of R-tree update operations, such as various node splitting and tree restructuring policies, as measured by query performance on the resulting tree. We use our model to study the performance of three well known R-tree packing algorithms. We show that ignoring buffer behavior and using number of nodes accessed as a performance metric can lead to incorrect conclusions, not only quantitatively, but also qualitatively. In addition, we consider the problem of how many levels of the R-tree should be pinned in the buffer.

1 Introduction

R-trees [3] are a common indexing technique for spatial data and are widely used in spatial and multidimensional databases. Typical applications include computer-aided design, geographic information systems, computer vision and robotics, multi-keyed indexing for traditional databases, temporal databases, and scientific databases.

A significant amount of activity has gone into developing better R-tree construction algorithms [10, 1, 9, 4, 5]. Most of this work has focused on proposing and comparing a new algorithm to previous ones, but little work has been done on methodology for comparing algorithms. Notable exceptions are the work of Kamel and Faloutsos [4], Theodoridis and Sellis [11], and Pagel et al. [8].

In [4, 8] the authors develop an analytical model for prediction of query performance. The model provides good insight into the problem, especially by establishing a quantitative relationship between performance and the total area and perimeter of the minimum bounding rectangles (MBRs) of the tree nodes, but suffers from one major drawback: the primary objective function for comparison is the number of nodes visited. In real databases some portion of the tree is buffered in main memory. This buffering of portions of the tree can significantly affect performance. Consequently, we propose that a better metric is the average number of disk accesses required to satisfy a query. The work of Theodoridis and Sellis [11] provides a fully analytical model not requiring the minimum bounding rectangles of the R-tree nodes as an input. This work also focuses on the number of nodes visited, but our model could be coupled with their model as we have done with the model of Kamel and Faloutsos.

In order to derive a model based on the number of disk accesses we borrow techniques developed by Bhide *et al* [2] (in the context of modeling databases with uniform access within each of several partitions) and develop a new buffer model. Our model can be used to evaluate the quality of any R-tree update operation, such as node splitting policies [3] or packing algorithms [4, 7, 9] as measured by query performance of the resulting tree. The model is very accurate and simple to understand, making it easy for researchers to integrate it into their studies. Furthermore, it is not only applicable to R-trees, but can easily be modified to model B-tree performance.

The main contributions of this paper are: first, the buffer model methodology; second, a demonstration of the importance of considering a buffer, third, insight into how large a buffer should be used, and fourth, showing the effect of pinning top levels of the R-tree into the buffer. We emphasize that the inclusion of a buffer not only changes the quantitative performance of the algorithms, but in some cases changes the qualitative ordering of the algorithms.

*This work is supported in part by NSF grant IRI-9610240 and Colorado Advance Software Institute grant number TT-97-05.

Our buffer model is an analytic model based on simple probability theory, but our overall performance model is a hybrid model. We start by using a packing algorithm to create actual R-trees for various data sets. Then, we compute the minimum bounding rectangles of tree nodes and use these as input to our buffer model.

The rest of the paper is organized as follows. In Section 2 we provide background information on R-trees and describe three loading algorithms that were chosen to test our results. Since this choice is by no means exhaustive, we emphasize that the purpose of this paper is not to draw irrevocable conclusions about the best loading algorithm (although it allows comparisons between the chosen ones), but rather to demonstrate the need and utility of the buffer model. In Section 3 we present the query performance models of Kamel and Faloutsos, our modification of the region query model, and our new buffer model. In Section 4 we present the validation of our model. Section 5 contains results from our model experiments and Section 6 concludes.

2 R-tree Algorithms

Due to space considerations we assume the reader is familiar with R-trees. The reader interested in a more information on R-trees is referred to the articles listed in the bibliography.

We consider three methods for R-tree creation:

Tuple-At-a-Time (TAT): This algorithm just inserts one tuple at a time into the R-tree using the *linear split* heuristic of Guttman [3].

Nearest-X (NX): This algorithm was proposed in [9]. The rectangles are sorted by x-coordinate. We assume that the x-coordinate of the rectangle's center is used. The rectangles are then packed into the nodes using this ordering. The tree is built bottom up with each subsequent level sorted on the x-coordinate.

Hilbert Sort (HS): This fractal based algorithm was proposed in [4]. The center points of the rectangles are sorted based on their distance from the origin as measured along the Hilbert Curve. Once again the tree is built bottom up.

3 Model Description

Kamel and Faloutsos [4] introduced an analytical model for computing the average response time of a query as a function of the geometric characteristics of the R-tree. Their model does not consider the effect of a buffer and performance is measured by the number of R-tree nodes accessed. In practice, query performance is mainly affected by the time required to retrieve nodes touched by the query which do not reside in the buffer, as the CPU time required to retrieve and process buffer resident nodes is usually negligible. We start by describing the model of Kamel and Faloutsos. We then modify this model to better fit our definition for uniformly distributed queries of a fixed size. Finally, we extend the model to take into account the existence of a buffer, including how to handle the pinning in the buffer of the top few levels of the R-tree.

In this section we consider a 2-dimensional data set consisting of rectangles to be stored in an R-tree T with $H+1$ levels, labeled 0 through H. All input rectangles have been normalized to fit within the unit square $U = [0,1] \times [0,1]$. We assume that queries are rectangles Q of size $q_x \times q_y$ uniformly distributed over the unit square. Note that a point query corresponds to the case $q_x = q_y = 0$. Our description concentrates on 2D. Generalizations to higher dimensions are straightforward.

Throughout this section we will use the following notation:

$$
\begin{aligned}
M_i &= \text{number of pages at the } i\text{th level of } T \\
M &= \text{Total number pages in } T\text{, i.e., } \textstyle\sum_{i=0}^{H} M_i \\
R_{ij} &= j\text{th rectangle at the } i\text{th level of } T \\
A_{ij} &= \text{area of } R_{ij} \\
A_{ij}^{Q} &= \text{probability } R_{ij} \text{ is accessed by query } Q \\
B_{ij} &= \text{number of accesses to } R_{ij} \\
\mathcal{A} &= \text{Sum of the areas of all MBRs in } T \\
L_x &= \text{Sum of the x-extents of all MBRs in } T \\
L_y &= \text{Sum of the y-extents of all MBRs in } T \\
N &= \text{number of queries performed so far} \\
N^* &= \text{expected number of queries required to} \\
&\quad\ \text{fill the buffer} \\
B &= \text{buffer size} \\
D(N) &= \text{number of distinct pages (at all levels)} \\
&\quad\ \text{accessed in } N \text{ queries} \\
E_T^P(q_x, q_y) &= \text{expected number of pages (buffer resident} \\
&\quad\ \text{or not) of } T \text{ accessed while performing} \\
&\quad\ \text{a query of size } q_x \times q_y \\
E_T^D(q_x, q_y) &= \text{expected number of disk accesses while}
\end{aligned}
$$

performing a query of size $q_x \times q_y$

Kamel and Faloutsos consider a bufferless model where performance is measured by number of nodes accessed (independent of whether they currently reside in the buffer). They observe that for uniform point queries the probability of accessing R_{ij} is just the area of R_{ij}, namely, A_{ij}. They point out that the level of T in which R_{ij} resides is immaterial as all rectangles containing Q (and only those) must be retrieved. Accordingly, for a point query the expected number of nodes retrieved as derived by Kamel and Faloutsos is[1]

$$E_T^P(0,0) = \sum_{i=0}^{H}\sum_{j=1}^{M_i} A_{ij} = \mathcal{A} \quad (1)$$

which is the sum of the areas of all rectangles (both leaf level MBRs as well as MBRs of internal nodes).

For region queries they show that the expected number of nodes accessed is [4]:

$$E_T^P(q_x, q_y) = \mathcal{A} + q_x L_y + q_y L_x + M q_x q_y \quad (2)$$

We have modified the model of Kamel and Faloutsos to handle query windows that fall outside the data space and data rectangle close to the boundary of the data space as suggested by Six et al [8]. Specifically:

1. For uniformly distributed rectangular queries of size $q_x \times q_y$ the top right corner of the query region Q, cannot be an arbitrary point inside the unit square if the entire query region is to fit within the unit square. Rather, Q_{tr} must be inside the box $U' = [q_x, 1] \times [q_y, 1]$.
2. The probability of accessing a rectangle $R = \langle (a,b),(c,d)\rangle$ is *not* the area of $R' = \langle (a,b),(c+q_x, d+q_y)\rangle$ as this value can be bigger than one. Rather, the access probability is the percentage of U' covered by the rectangle $R' \cap U'$.

Thus, we change the probability of accessing rectangle i of level j to:

$$A_{i,j}^Q = \frac{\text{area}(R' \cap U')}{\text{area}(U')} \quad (3)$$

$$= \frac{C * D}{(1-q_x)(1-q_y)} \quad (4)$$

[1] We have modified the notation of [4] to make it consistent with the notation used in this paper.

where $C = [\min(1, c+q_x) - \max(a, q_x)]$ and $D = [\min(1, d+q_y) - \max(b, q_y)]$. Further details can be found in [6] and are omitted for brevity. We use this $A_{i,j}^Q$ in all subsequent discussion and results.

3.1 A Buffer Model

Bhide *et al* [2] analyze the LRU buffer replacement policy for databases consisting of a number of partitions with uniform page access within each partition. While modeling performance during buffer warm-up they observed that the buffer hit probability at the end of the warm-up period is a good estimator of the steady state buffer hit probability.

We conjectured that similar behavior would occur in the context of R-trees when applying the LRU replacement policy, namely, that the steady state buffer hit probability is virtually the same as the buffer hit probability when the buffer *first* becomes full. Our conjecture was verified experimentally as discussed in Section 4.

Under uniformly distributed point queries, the probability of accessing rectangle R_{ij} while performing a query is A_{ij}. Accordingly, the probability that R_{ij} is not accessed during the next N queries is $P[B_{ij} = 0|N] = (1-A_{ij})^N$. Thus, $P[B_{ij} \geq 1|N] = 1-(1-A_{ij})^N$ and the expected number of *distinct* pages accessed in N queries is,

$$D(N) = \sum_{i=0}^{H}\sum_{j=1}^{M_i} P[B_{ij} \geq 1|N] = M - \sum_{i=0}^{H}\sum_{j=1}^{M_i}(1-A_{ij})^N \quad (5)$$

Note that $D(0) = 0 < B$ and $D(1) = \mathcal{A}$ (which may or may not be bigger than B). The buffer, which is initially empty, first becomes full after performing N^* queries, where N^* is the smallest integer that satisfies $D(N^*) \geq B$. The value of N^* can be determined by a simple binary search:

```
ComputeN*()
    high ← 1;
    while D(high) < B do
        high ← 2 * high;
    low ← high/2;
    N* ← ⌈(low + high)/2⌉;
    while D(N*) < B or D(N* − 1) ≥ B do
        if (D(N*) < B)
            low ← N*;
        else
            high ← N*;
        N* ← ⌈(low + high)/2⌉;
    return N*;
end
```

Now, while the buffer is not full the probability that R_{ij} is in the buffer is equal to $P[B_{ij} \geq 1]$. The probability that a random query requires a disk access for R_{ij} is $A_{ij} \cdot P[B_{ij} = 0]$.

Since the steady state buffer hit probability is approximately the same as the buffer hit probability after N^* queries, the expected number of disk accesses for a point query at steady state is

$$\sum_{i=0}^{H} \sum_{j=1}^{M_i} A_{ij} \cdot P[B_{ij} = 0 | N^*] = \sum_{i=0}^{H} \sum_{j=1}^{M_i} A_{ij} \cdot (1 - A_{ij})^{N^*} \tag{6}$$

The above derivation also holds for region queries provided that A_{ij}^Q is used instead of A_{ij}, i.e.:

$$E_T^D(q_x, q_y) = \sum_{i=0}^{H} \sum_{j=1}^{M_i} A_{ij}^Q \cdot (1 - A_{ij}^Q)^{N^*}$$

Finally, we point out that it is easy to extend the above results to model a buffer management policy that pins the top few levels of the R-tree in the buffer: simply reduce the number of buffer pages by the number of pages in these pinned levels and omit the top levels from the model.

4 Model Validation

We validated the model by comparing it with simulation. The simulation models an LRU buffer and, like the model, takes as input the list of the MBRs for all nodes at all levels. It then generates random point queries in the unit square and checks each node's MBR to see if it contains the point. If the MBR does contain the point the node is requested from the buffer pool. If the node is not in the buffer pool, the least recently used node in the buffer is pushed out and the new node put on the top of the LRU stack. Note that the simulation is accurate in the sense that an R-tree implementation retrieves all and only those rectangles (internal or not) that intersect the region query. Confidence intervals were collected using batch means with 20 batches of 1,000,000 queries each, resulting in confidence intervals of less than 3% at a 90% confidence level.

We ran comparisons for three different R-trees and 6 different buffer sizes for each tree. Each of the R-trees has 1668 nodes, but the MBRs of the nodes are different, as produced by the three different packing algorithms. In Table 1 we report the average number of pages required from disk per point query as predicted by the simulation, model, and the percent difference (relative to the simulation), for two of the trees. The third has similar comparisons. All of the results are within 2%, which is less than the confidence intervals returned from the simulation. Other comparison experiments not shown resulted in agreement within 2% or less. Simulation of region queries gave similar results.

	Rtree-1			Rtree-2		
bufsize	simul	model	% dif	simul	model	% dif
50	2.3120	2.2682	1.189	2.2060	2.1651	1.185
100	1.7008	1.6862	0.86	1.6288	1.6166	0.75
200	1.1627	1.1599	0.24	1.1194	1.1172	0.20
400	0.9347	0.9340	0.07	0.9009	0.9007	0.02
1000	0.4701	0.4700	0.02	0.4534	0.4532	0.04
1400	0.1770	0.1769	0.06	0.1714	0.1712	0.12

Table 1: Validation: average number of disk accesses per query for model and simulation.

5 Model Results

In this section we present performance results as predicted by our model. We consider two types of queries: point queries and region queries. A point query specifies a point in the unit square and finds all rectangles that contain the point.

Region queries specify a region of a given size and find all rectangles that intersect with this region. Each query is a rectangle of size $q_x \times q_y$ whose upper right corner is contained in region U' as described in Section 3.

5.1 Need For Consideration of Buffer Impact

Figure 1 plots the number of disk accesses versus buffer size for the Long Beach data set extracted from the TIGER system of the U. S. Bureau of Census, assuming R-trees with 100 rectangles per node. The top plot is for point queries, the bottom for region queries. Consider the top two curves of the bottom plot. For small buffer sizes the TAT algorithm requires fewer disk access than the NX algorithm. At a buffer size of 200 the performance of the two algorithms crosses and the NX algorithm becomes the better algorithm. Hence, ignoring buffering would result in the incorrect conclusion that TAT is better than NX, thus underscoring the importance of including buffer effects in comparison studies.

A second experiment shows that not considering the buffer impact can introduce significant errors. In

Figure 2 we present results for synthetic data. Uniformly distributed data sets were created containing between 10,000 and 300,000 rectangles. For each rectangle the lower left corner was uniformly distributed over the unit square. All rectangles are squares, where the length is chosen uniformly between 0 and ϵ. The value of ϵ is fixed for all data set sizes and is equal to $2\sqrt{0.25/10000}$. Thus, for a 10,000 rectangle data set the sum of the rectangle areas is equal to 0.25 of the unit square, for 100,000 rectangles the total area equals 2.5 times the unit square. This is similar to the experimental methodology used in [4].

The top graph plots the number of nodes visited (i.e. no buffering considered) versus the number of rectangles in the data set. The middle graph plots the number of disk accesses versus data set size for a buffer size of 10, and the bottom graph for a buffer size of 300. Ignoring buffer impact (top) leads to the conclusion that querying an R-tree of 300,000 rectangles is no more expensive than querying an R-tree of 25,000 rectangles. This could cause a query optimizer to produce a poor query plan. Once buffering is considered, the fact that larger trees are more expensive is evident.

5.2 Choosing Buffer Size for R-trees

In this section we study the reduction in disk accesses obtained from increasing the buffer size. Since main memory is a valuable resource, some insight into the gains from allocating more buffer space is needed. Consider Figure 1 again where the number of disk accesses versus buffer size is plotted for both point and region queries run against the Long Beach tiger data set. The data set has 53,145 rectangles (actually line segments). A node size of 100 rectangles is assumed resulting in the HS and NX algorithms having 532 pages at the leaf level, 6 pages at level 1, and 1 root node. Buffer size is varied from 2 to 500 pages (from 0.4% to 92.8% of the R-tree).

The top plot is for point queries. The curves plotted from top to bottom are TAT, NX, and HS. The R-tree produced by the TAT algorithm is poorly structured, and, as a result, seems to benefit significantly from each increase in buffer size. The HS tree is much better structured and, while experiencing a halving of the number of required disk accesses for a buffer size of 10%, additional buffer increases only help modestly. The NX algorithm does not experience as much of a reduction in needed disk accesses for small buffer sizes as the HS algorithm. We hypothesize that the better the R-tree structure, the more capable it is to capitalize on small amounts of buffer for point queries, whereas the worse the R-tree structure, the more linear the reduction in disk accesses.

data size	level 1 (root)	level 2	level 3	level 4
40000	1	3	64	1600
60000	1	4	96	2400
80000	1	6	128	3200
100000	1	7	160	4000
120000	1	8	192	4800
150000	1	10	240	6000
200000	1	13	320	8000
250000	1	16	400	10000

Table 2: Number of nodes per level

The bottom plot is for 1% region queries. Note that for region queries none of the curves have a well defined "knee". Thus, based on this experiment it appears it is possible that when executing region queries reductions in disk accesses for increasing the buffer size is more linear than for point queries.

5.3 Choosing the Number of Levels to Be Pinned

Past buffer management studies have shown that for B-trees the root and maybe the first level should be pinned in the buffer pool. Pinning nodes decreases the buffer size remaining for other pages, but guarantees that the pinned pages will be in buffer. We present experiments to investigate what gains in performance can be expected from pinning R-tree nodes, and how many levels should be pinned.

For the experiments in this section we wanted to get deeper R-trees. To do this we created synthetic data sets with 40,000 - 250,000 rectangles and used R-tree nodes of size 25. This resulted in 4 level R-trees with the numbers of nodes at each level being as shown in Table 2.

In Figure 3 we plot the number of disk accesses for point queries versus the data size for three buffer sizes assuming the point queries on trees created by the HS algorithm. Results for the other algorithms are similar. The plot on the top is for a buffer of 500 pages, middle 1000 pages, and the bottom graph is for a buffer size of 2000 pages. The number of disk access for not pinning any levels, pinning the first (root) level, and pinning the first two levels is the same. All three scenarios are plotted as the single top line in all three graphs. The bottom line is for pinning the first three levels.

As a general rule of thumb, pinning a level makes a difference if the total number of pages pinned is within a factor of 2 of the buffer size, but when the total number of pages pinned is less than $\frac{1}{3}^{rd}$ of the buffer size only marginal benefits are seen. This is because for a smaller relative number of pinned pages the LRU replacement policy succeeds in keeping the top levels in the buffer without explicit pinning. For example, for a buffer size of 500 and a data size of 250,000 rectangles, pinning three levels pins 417 pages. This results in 53% fewer disk accesses. When the data size is 80,000 rectangle 135 pages are pinned and the saving is only 4%. For a buffer size of 2000, pinning of the the first 3 levels makes almost no difference since the number of pages pinned is less than $\frac{1}{4}^{th}$ of the buffer size.

Thus, for point queries pinning levels is advantageous, but only when the total number of nodes pinned is within a factor of 2 of the buffer size. Note, pinning never hurts performance and hence for a fixed size buffer dedicated to an R-tree application as many levels as possible should be pinned. If the buffer is shared among many applications, the benefit of pinning must be compared to the other applications. Our buffer model can be used to predict the benefit accrued by pinning R-tree levels.

For region queries all experiments to date (not shown) have resulted in only a modest (2-5%) improvement from pinning levels. Thus, it appears that pinning may mostly benefit point queries, and even then only under special scenarios.

6 Conclusions

We have developed a new buffer model to predict the number of disk accesses required per query for a given input R-tree (specified by the minimum bounding rectangles of each node in the tree). Our model has been shown experimentally to agree with simulation within 2% for numerous test cases. The model is both simple to implement and quick to solve, thus providing a useful methodology for further studies.

With our model, we have demonstrated that using the number of nodes accessed per query as a performance metric is not sufficient, because it ignores buffer effects. We show that once buffer effects are taken into account not only does quantitative predicted performance change, but the qualitative predictions can change as well. In particular, the actual ordering of policies can differ when buffering is not considered versus when it is considered. Thus, it is essential to consider buffer effects in a performance study of R-trees and use number of disk accesses as the primary metric.

In addition, we used our model to determine that for the data considered, small amounts of buffer can super-linearly improve performance of well structured R-trees for point queries, but that poorly structured trees see only a more linear improvement in performance due to the increase in buffer space. In addition, we find that for region queries even well structured R-trees see a more linear improvement as buffer is added.

Finally, we show that in most cases pinning the top levels of the R-tree has little effect on performance relative to just using an LRU buffer. For point queries we did find certain scenarios where if the buffer size is a about the same or a few times larger than the number of nodes at a level, pinning that level does significantly improve performance.

Acknowledgments.

We would like to thank Jeff Edgington for developing the R-tree packing algorithm code. We would also like to thank Ken Sevcik for useful discussions about a preliminary version of this work.

References

[1] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The $r^{\star}$-tree: an efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD*, pages 323–331, may 1990.

[2] A. Bhide, A. Dan, and D. Dias. A simple analysis of lru buffer replacemtn policy and its relationship to buffer warm-up transient. In *Proc. IEEE International Conference on Data Engineering (ICDE)*, 1993.

[3] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD*, pages 47–57, Boston, Mass, June 1984.

[4] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. In *Proc. of Second Int. Conf. on Information and Knowledge Management (CIKM)*, Washington, D.C., November 1993.

[5] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: an improved r-tree using fractals. In *Proc. of VLDB Conference,*, pages 500–509, Santiago, Chile, September 1994.

[6] S.T. Leutenegger and M.A. Lopez. The effect of buffering on the performance of

r-trees. Technical Report COMP 96-02, Mathematics and Computer Science Department, University of Denver, 1996. URL: www.cs.du.edu/users/leut/RtreeBufferTR.ps.

[7] S.T. Leutenegger, M.A. Lopez, and J.M. Edgington. Str: A simple and efficient algorithm for r-tree packing. In *Proc. of the 14th International Conference on Data Engineering (ICDE 97)*, pages 497–506, 1997.

[8] B-U. Pagel, H-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. ACM PODS*, pages 214–221, may 1993.

[9] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using packed r-trees. In *Proc. ACM SIGMOD*, Austin, Texas, May 1985.

[10] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+ tree: A dynamic index for multidimensional objects. In *Proc. of 13th International Conference on Very Large Databases (VLDB-87)*, pages 507–518, sep 1987.

[11] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. In *Proc. of 8th Symposium on Principles of Database Systems (PODS)*, May 1996.

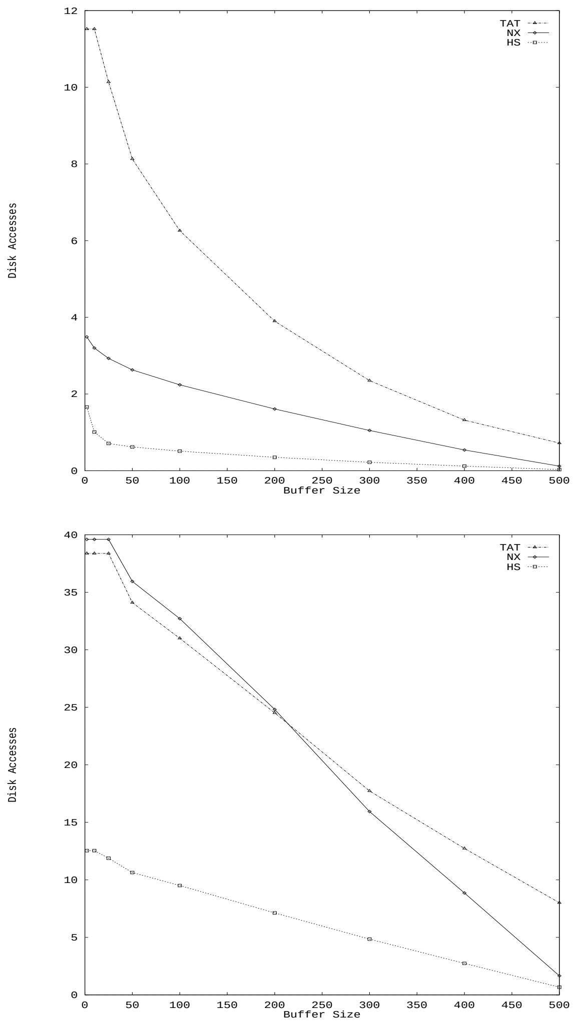

Figure 1: Sensitivity to Buffer Size, Long Beach Data: Top: Point Queries; Bottom: 1% Region Queries

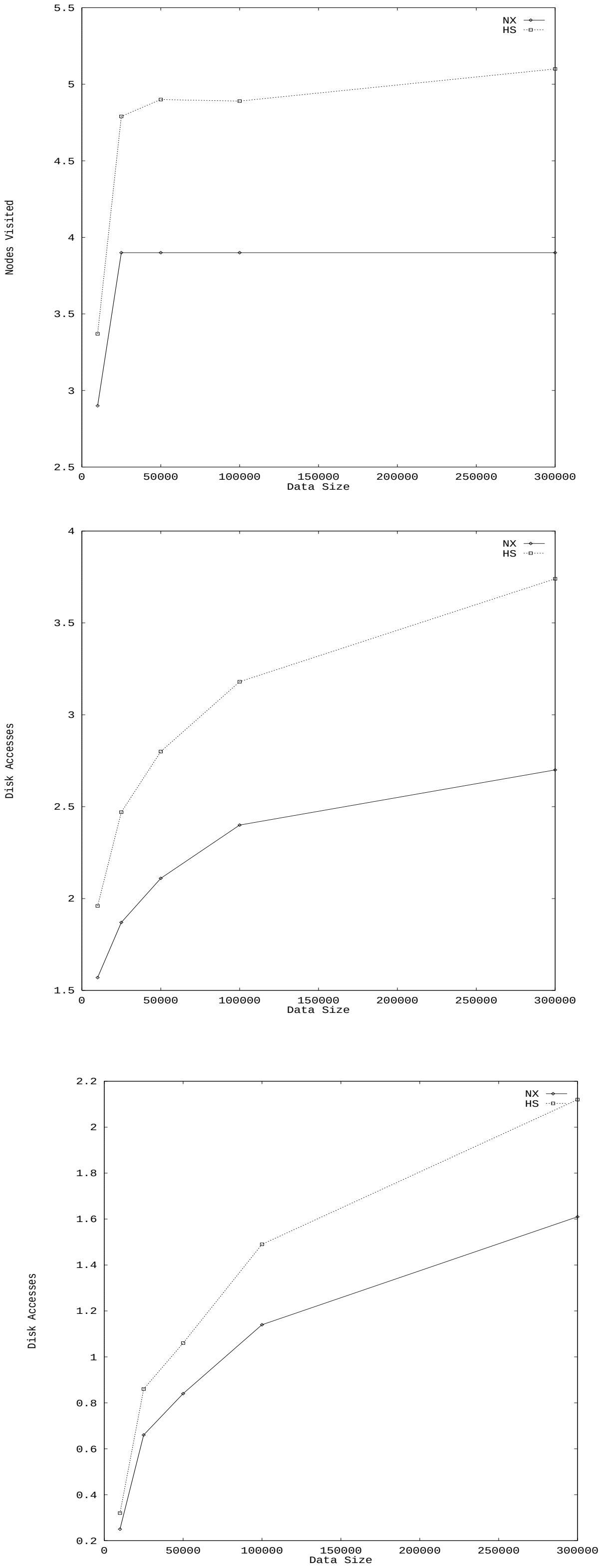

Figure 2: Disk Accesses versus Data Size for NX and HS: Top: NoBuffer ; Middle: Buffer Size = 10 ; Bottom: Buffer Size = 300

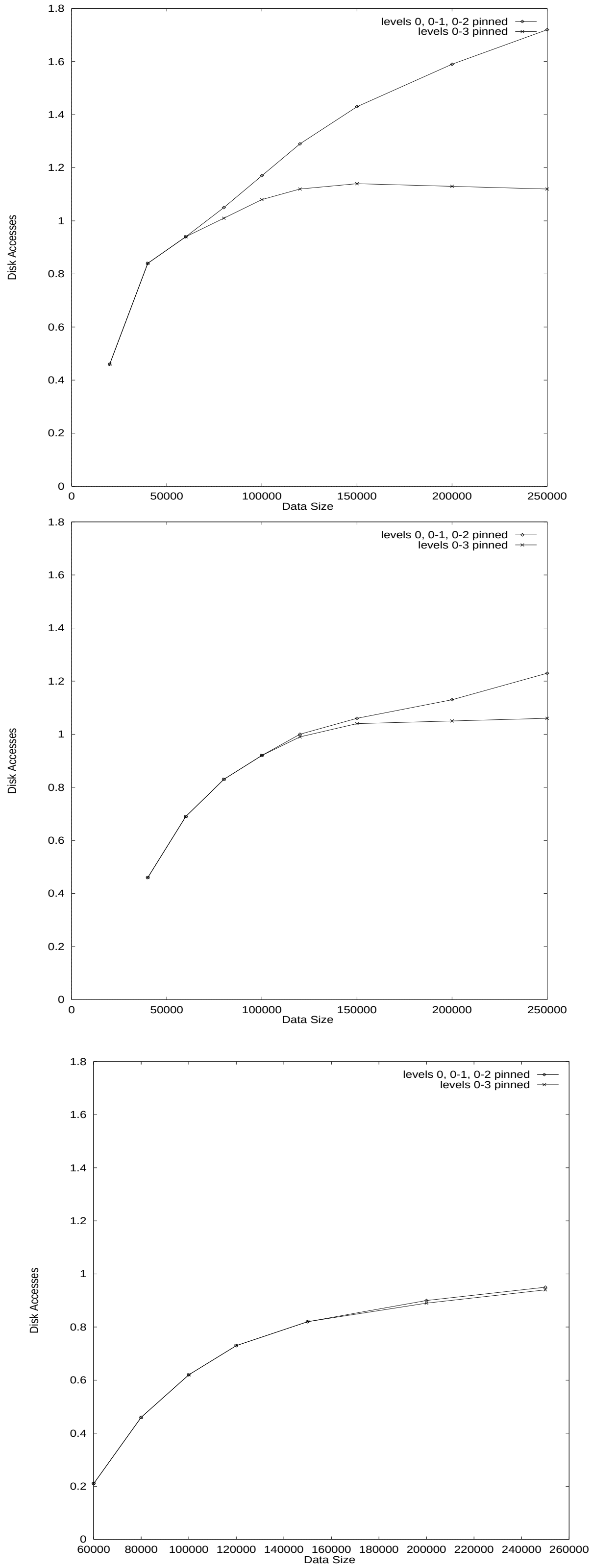

Figure 3: Effect of Pinning, Disk Accesses versus Data Size for HS: Top: Buffer Size = 500 ; Middle: Buffer Size = 1000 ; Bottom: Buffer Size = 2000