World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:11, 2008

MONPAR - A Page Replacement Algorithm for a Spatiotemporal Database

U. Kalay, and O. Kalıpsız

***Abstract*—**For a spatiotemporal database management system, I/O cost of queries and other operations is an important performance criterion. In order to optimize this cost, an intense research on designing robust index structures has been done in the past decade. With these major considerations, there are still other design issues that deserve addressing due to their direct impact on the I/O cost. Having said this, an efficient buffer management strategy plays a key role on reducing redundant disk access. In this paper, we proposed an efficient buffer strategy for a spatiotemporal database index structure, specifically indexing objects moving over a network of roads. The proposed strategy, namely *MONPAR*, is based on the data type (i.e. spatiotemporal data) and the structure of the index structure. For the purpose of an experimental evaluation, we set up a simulation environment that counts the number of disk accesses while executing a number of spatiotemporal range-queries over the index. We reiterated simulations with query sets with different distributions, such as uniform query distribution and skewed query distribution. Based on the comparison of our strategy with well-known page-replacement techniques, like LRU-based and Priority-based buffers, we conclude that *MONPAR* behaves better than its competitors for small and medium size buffers under all used query-distributions.

***Keywords*—**Buffer Management, Spatiotemporal databases.

I. Introduction

CONSIDERING a real-time query execution, execution time is known as the typical parameter that reveals the quality of system performance. Due to the ever-increasing gap between average main-memory access time and average secondary-storage access time, I/O cost is still the dominant factor for the optimization of the execution time. To alleviate this burden, indexes, associated with database files, have been employed to organize the hard disk pages in such a way as to minimize the number of disk accesses during query execution or other database operations. While utilizing indexes is indispensable in today's database systems, there are still other means of reducing I/O cost such as having an efficient buffer management, applying effective query optimization techniques and better data clustering techniques within disk pages. Each of these research directions have been addressed in the literature with their novel contributions. Moreover, the inevitable affect of recent advances in multi-dimensional databases calls for innovations in these areas aiming at a better execution time while minimizing the I/O cost. Multidimensional indexing, for instance, is extended on the existing indexing methods with additional data structures to accommodate the specific requirements of new applications. Accordingly, with changing indexing strategies, the need for more robust solutions arises naturally in other related areas mentioned previously.

In this paper, a novel buffer management strategy, namely *MONPAR*, is designed for a moving object index structure, namely MON-tree. Since MON-tree is a spatiotemporal index structure designed for objects moving over a network of roads, it consequently needs a specialized buffer management aiming at reducing the number of disk accesses. In the next section, we describe main aspects of some well-known buffer management strategies and explain our motivation for a new one. Next, in the 3rd section, we examine the main features of the MON-tree index structure, on which we built up the new buffer management algorithm, *MONPAR*. Lastly, we describe our simulation model in section 4 and evaluate the performance of our algorithm based on the comparisons with other buffer models.

Manuscript received August 31, 2006.

U. Kalay is with Dept. of Computer Engineering, Yildiz Technical University, Istanbul, Turkiye (e-mail: utku@ce.yildiz.edu.tr).

O. Kalıpsız is with Dept. of Computer Engineering, Yildiz Technical University, Istanbul, Turkiye (e-mail: kalipsiz@yildiz.edu.tr).

II. Buffer Management

Buffer is a small part of main memory allocated for the purpose of keeping the hard disk pages that is expected to be used soon. While the idea of buffering is a traditional operating system concept, it has an important impact on the performance of index structures as well. Basically, while executing a query, pages requested by the query are supposed to be read from hard disk and to be located in the buffer area in order to serve for anticipated disk accesses without accessing the disk redundantly. At this point, if the buffer is unable to locate all of the target pages due to its limited capacity, an algorithm aiming at keeping the most "important" pages in the buffer area is needed. In other words, the algorithm, called as page replacement algorithm, is expected to have the ability to select the best appropriate page (called as victim) to drop from the buffer in order to make room for the new requested page. It is not always possible to select the best victim page for the replacement. Nevertheless, many adequate traditional solutions exist in the literature, some of which are LRU (***L***east ***R***ecently ***U***sed), NRU (***N***ot ***R***ecently ***U***sed), and LFU (***L***east ***F***requently ***U***sed). These algorithms are originally designed based on the patterns of disk page usage in general

manner and do not always fit well into the database environment. For example, LRU page replacement algorithm replaces the page that has not been accessed for the longest time. LRU gives the highest priority to the last referenced page by keeping it in the buffer until all other pages in the buffer are replaced or referenced again. Hence, although simple to implement, LRU is unable to differentiate the pages that have frequent reference from the pages that have infrequent reference [1]. Additionally, in case of tree-based database index structures, the position of the page in the structure can be a valid criterion for the victim selection, thus LRU is inappropriate due to the lack of this knowledge.

In order to adapt the LRU replacement strategy to database applications, *priority-based LRU strategy (LRU-P)* was suggested in [2]. In this strategy, assuming a tree-based spatial access method used, priority of a page in an index depends on its level in the tree. While the root has the priority level equal to the height of the tree, the pages at the lower levels have priority levels corresponding to their distance from the leaves. It is important to note that yet *LRU-P* selects the least recently used page as victim, but it selects the victim among the pages in the buffer having the lowest priority level. Although it is a good idea to apply this kind of priority assigning to each level, LRU-P strategy considers only the position of page in the tree.

All derivatives of the LRU method, like *LRU-P* [2] and *O*LRU, *I*LRU [3] and all other traditional solutions (NRU,LFU, LRU-K, clock page.) does not have any knowledge about the type or content of the stored pages. However, in multidimensional databases such as spatial databases, it would be better to analyze the content of the page in order to select the best victim page in the buffer. For this purpose, spatial page replacement algorithm (*Spatial*PageReplacement) has been proposed in [2]. In essence, this algorithm requires a function *SC (p)* computing the area of MBR containing all the entries of the page. The algorithm selects the victim page, *p* by applying the function *SC* as follows:

$$\left\{ \; p \mid p \in \text{buffer} \wedge (q \in \text{buffer} \Rightarrow \; \text{SC}(p) \leq \text{SC}(q)) \; \right\} \quad (1)$$

The experiments in [2] shows that it is not advisable to use the pure spatial page replacement algorithm for some query distributions. Lastly, an adaptable solution combining LRU and spatial page-replacement algorithms was investigated in [2] in order to achieve a robust organization. Unfortunately, the findings from [2] is valid only spatial index like R*tree. We think that spatial page-replacement would be a starting point to design specialized buffer organizations for other multi-dimensional indexes such as moving object index, MON-tree. For this purpose, we developed *MONPAR* on the basis of *Spatial*PageReplacement.

III. A Buffer Organization for Mon-Tree

A. MON-Tree

MON-tree by Almeida and Güting [4] is an efficient organization of a group of R-tree, which is a widely used spatial index structure. R-tree spatial index structure [5], similar to traditional B-tree, is an excellent index structure for query-based static systems. MON-tree is essentially designed for the purpose of indexing the past movement of objects traveling over a network of roads. Fig. 1 shows its basic architectural structure.

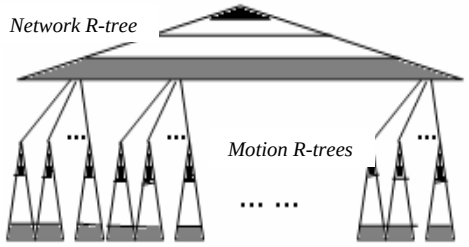

Fig. 1 MON-tree structural overview

MON-tree is a 2-level organization including a set of 2D R-trees and a hash table (which is not shown in figure). While the R-tree at the upper level indexes the edges of the network that had traffic on it, each of R-tree at the lower level stores the movements along the corresponding edge. Thus, the number of low-level R-trees is equal to the total number of edges at leaves of the top level R-tree. In addition, hash table in main memory is an auxiliary structure to directly access the movements on a specific edge by hashing an edge, E_i to the corresponding low-level R-tree. An object movement is represented as a rectangle (p_1, p_2, t_1, t_2), which means an object motion starts at p_1 part of an edge, E_i at time t_1, and ends its movement at p_2 part of E_i at time t_2, while $0.0 \leq p_1 < 1.0$ and $0.0 < p_2 \leq 1.0$ are hold.

Now, consider a spatial-temporal range query, $Q\ (q_{spatial}, t1, t2)$, where $q_{spatial}$ is a traditional spatial range query that is valid for the duration from *t1* to *t2*. At the first step, $q_{spatial}$ is the static range query over the top-level R-tree that finds the edges that are covered by query region. By accessing an edge stored in the top-level, we get the corresponding low-level R-tree that stores the past movements on this edge. In fact, since the edge may partially be within the query region, the covered parts of the edge are selected in main memory with a minimal execution overhead. Therefore, the output from the first step is the set of low-level R-trees and a query set that contains the part of edges covered by the query region, ***w***. Then, the second step is to find the moving objects of which past movement regions are intersecting with any query region in ***w.*** That is actually done by another range query executions over the low-level R-trees found in the first step. Detailed explanation on this range query execution and performance improvements can be found in [4] for more interested readers. We implemented MON-tree index structure as we have described in [6].

B. MONPAR

As we noted in section 2, *Spatial*PageReplacement is a specialized algorithm for R-tree index structure. Since MON-

tree has been designed as a group of 2D R-trees, we think that it is worth to mention another specialized buffer algorithm which applies the idea in the *Spatial*PageReplacement with the structural characteristics of MON-tree. Based on these motivations, we implemented the *MONPAR* algorithm from the pseudo-code shown below.

```
r : requested page
H: set of headers page
N: set of  network pages
M: set of  motion pages
C : buffer capacity
H=N=M= ø
SC(p): area(mbr(p)) // area of mbr of a page

MONPAR (r ){
   if (‘r’ is in buffer)
      update ‘r’s statistics if required
   else
      type := the type of page ‘r’
      if (|H|+|N|+|M| < C)
            add ‘r’ into corresponding set
      else if(|H|+|N|+|M| == C)
       {
          victim=null;
           if (type = = network)
                  victim := {v| v ∈ N, p ∈ N, SC(v)<SC(p)}
            else if (type == motion)
                  victim := {v| v ∈ M, p ∈ M, SC(v)<SC(p)}
           else
                  victim := LRU(v| v ∈ H)
           write victim page to disk
           add ‘r’ into corresponding set
       }
}
```

3 types of pages can be designed from the structural overview of the MON-tree which is depicted in Fig. 1. Two of them, network pages (*N*) and motion pages (*M*), are easy to refer when we look at the figure. Additionally, we needed a third set containing header pages (*H*). Considering long-term motion simulations, a high number, if not all, of header pages are expected to accommodate in the buffer due to the fact that each edge having traffic on it is represented with an additional R-tree. In our experiments, header page actually holds the configuration information for the corresponding R-tree that naturally results in internal fragmentation. It is a good idea to collect all configuration information throughout the MON-tree within a couple of disk pages. While this scheme leads to more complicated organization for the overall tree configuration, this would eventually eliminate the set *H* in our *MONPAR* algorithm. In that case, we would keep the header pages in the buffer all the time since there are a few of them. As a future work, we plan to discard the set *H* in our further experiments after some minor reorganization.

In the algorithm, SC (*p*) (spatial criteria for *p*), as we noted beforehand, is the function calculating the total area of MBR of the entries in the page, *p*. Since SC results for different set of pages (*N* and *M*) have different value ranks, they are not comparable. As expected, the replacement already happens between the pages of the same type.

According to the pseudo code, if the requested page *r* is in the buffer, there is no further operation to execute except keeping some statistics only for LRU, which is the case if set *H* is organized as a LRU buffer. If *r* is not in the buffer and the buffer capacity, *C* will not be exceeded, the page is added into the corresponding set. Otherwise, if *C* is already at its maximum, a victim should be determined from the set having the same type of *r*. At this point, the comparisons done in the selection operation is the same as the *Spatial*PageReplacement. Once the most appropriate page (which has the minimum *mbr* area) has been determined, it is replaced with the requested page, *r*.

It is important to note that, in our algorithm, once the buffer becomes full (|H|+|N|+|M| = C), the size of each set in the buffer would not change afterwards. This may seem to be a contradiction with the adaptive solutions in [2], however, in our structure, usage statistics of each different set (*N,M,H*) naturally depends to each other. Thus, it is not applicable to apply a hybrid solution combining the strategies from LRU and *Spatial*PageReplacement as it is done in [2], which studies only a single R-tree. Moreover, we are dividing the buffer into three subsections, each of which has a fixed-size length determined by the initial query distribution conditions. Admittedly, we ignore the query distributions which cause instability between the numbers of the pages in the buffer. It is clear that this would lead to redundant replacements. Fortunately, we realized that for range queries on MON-tree it is rare to generate queries that modify the balance between the requests on each type of set. For example, if high number of spatial-temporal queries is generated for a specific range, that is, this would fill up the buffer with the pages containing the edges in this region; then this would lead the corresponding low-level motion R-tree pages to place into the buffer. In fact, the larger the query spatial region area is in the query, the higher number of motion R-tree pages is requested from hard disk. Therefore, the balance between the numbers of different type of pages requested for a query does not change dramatically due to the structural characteristics of MON-tree.

IV. Experiments

We used the same simulation environment as we did in [6]. Basically, we index the moving objects traveling over a network of roads and query on this dynamic dataset. Once the traffic generator generators the traffic over the predefined network, it is possible to execute many types of queries over this series of movements. We completed our studies on range queries about the past events.

Spatial index implementations in Java programming language, namely SaIL[7], has been the core for implementing MON-tree index structure and all buffer organizations. Additionally, we inspired from the work at [7] in order to generate network traffic obeying the normal distribution over the roads of the network. Fig. 2 depicts the basic modules and those that we integrated with in order to evaluate the buffer performance.

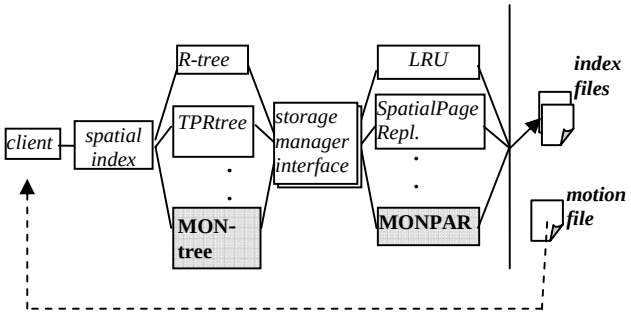

Fig. 2 Main modules of simulation environment

We implemented a discrete-time event-generation environment that randomly generates objects moving over the network based on the parameters, some of which is shown in Table I. Motion vectors of a group of objects at each time step is updated based on the initialization parameters, such as agility and maximum update interval *(mui)*. While the motion vector of each object at each time step is stored in a text file, the trajectory of each point is updated in the index structure. After completion of the generating data set (*DS*) and constructing the corresponding index with a predefined leaf capacity (*LF*), query processing module executes the queries in the query set (*QS*) on the index structure. For the sake of more comparability of results, before performing each new query set, the buffer was cleared.

Under these conditions, the established MON-tree index structure has a total of 8809 disk pages, including 1330 header pages, 241 network pages and the remaining 7238 motion pages that are distributed over 1329 motion R-trees. Each query set includes 100 spatiotemporal range queries over the road network between the simulation intervals, whereas each has spatial and temporal distribution that is different from other sets. We generated 3 types of sets: The first type, namely ***u***niform ***q***uery ***d***istribution (***uqd***), includes queries obeying the uniform distribution characteristics in both spatial and temporal dimensions. The second one, namely ***s***kewed ***q***uery ***d***istribution with *probability* of ***1.0***(***sqd-1)***, includes queries each of which spatially covers a randomly determined area (actually this selected area covers %20 of all road network region in each dimension) and has a randomly determined temporal interval of 10 time steps throughout the SL. In literature, this type of queries is known as hot-spot access queries. Lastly, the third type of queries, namely ***s***kewed ***q***uery ***d***istribution with probability of ***p*** (***sqd-p)***, is generated in order to evaluate the behavior of the buffer against the instantaneous variations of the query region on only spatial region dimensions. To do so, we deliberately generate a query out of the skewed region with a probability of ***p***, whereas the remaining queries still obey the ***sqd*** characteristics.

Each query set with the above characteristics is executed over MON-tree with a specific buffer organization. In our tests, we compared *MONPAR* with Random, LRU and Priority buffer organizations. As can be guessed, random buffer selects the victim in the buffer randomly, and LRU buffer selects the victim solely based on the LRU criterion. Lastly, Priority buffer, unlike LRU-P presented in section 2, solely requires the knowledge of requested page's position in the structure. In our implementation, for all R-tree pages, we assign the priority value to a page that equals to its distance from leaves. With *MONPAR*, not only is structural position analysis of the pages involved in selection criterion by categorizing them into 3 groups, but spatial characteristics are also involved in the decision criterion by applying appropriate calculations (like *SC*).

TABLE I
SOME OF THE SIMULATION PARAMETERS

Parameter	Meaning	Value(s)
PS	disk page capacity	4K
LC	leaf capacity	10
SL	simulation length	400 time steps
DS	dataset size	250 moving obj.
QS	query set size	100 spatiotemporal query
BC	buffer capacity	50, 100, 250, 500 pages

Our experiments are conducted with a time interval of *SL*=400 on the different-size page buffers in order to see the relation between the buffer size and the type of page replacement. The size of the buffer was chosen so that the buffer can hold 0.5%, 1%, 2.5%, 5% of overall index pages, which results in buffer sizes of 50, 100, 250 and 500 pages. The performance gain is given as the ratio of number of requests served from buffer to total requests in percent, during the execution of the query set.

After completion of a number of experiments with different query distributions, buffer sizes and buffer organizations, we end up with a list of figures depicted as in fig.3. First, we consider the buffer performance in relation to the buffer size. Generally speaking, according to the figures in fig.3, it is clear that buffer capacity is not always the main parameter that improves the performance of the buffer. For example, this is shown in the case with *uqd* distribution on buffers obeying Random and LRU rules. Here, the buffer size has nearly no affect on the performance. The reason is that under such a uniform distribution, buffers having no knowledge about the structure are easily filled up with the low levels of the index, which leads to redundant page faults. When the buffer has knowledge about the level of pages (as Priority and *MONPAR* do), the performance increases with the increasing buffer size. This observation shows the clear effect of using structure-aware buffers. Additionally, when other distributions are applied, especially sqd-*1*, the same buffers (Random and LRU) responses to the buffer size modifications accordingly. It is equally important to note that under the sqd-*1* distribution condition, as the buffer capacity increases, the performances of all type of buffers already increase rapidly to a peak value. As Fig. 3(b) shows, the LRU and Random buffer's performance catches the specialized buffers' performance at almost medium buffer-sizes. Eventually, at BS=500, all buffers can hold overall requested index nodes.

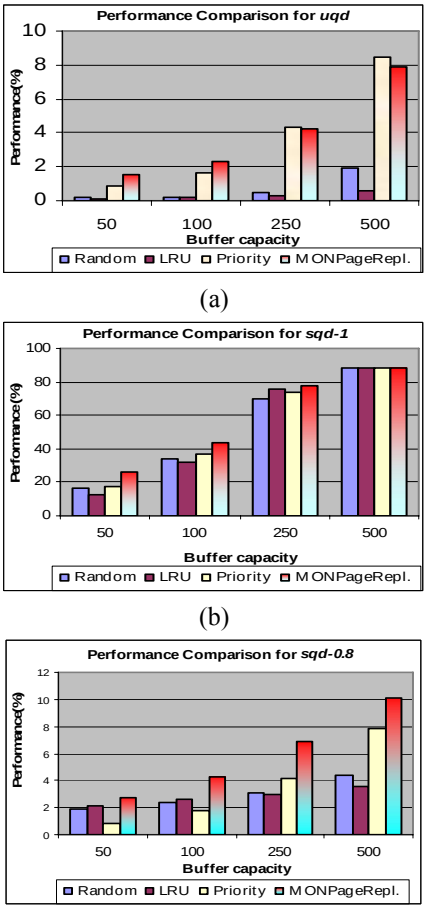

Fig. 3 Performance results depending on the buffer size and the buffer type for each query distribution

Now, take into consideration the performance gain when using specialized buffer organizations. Generally speaking, there appear to be a competition between Priority and *MONPAR* buffers. For the test series including *uqd*-type queries, the performance of *MONPAR* beats the others, especially for the buffer sizes of less than about 200. The reason is that, for the large buffers (above capacity of 250), under such a totally random query distribution, the initial distribution of N,M and H sets plays a crucial role for the buffer's later behaviors. Of course, if we reduce the random behavior of queries as we did in the other distributions; we observe the alleviation of the dominant affect of initial set capacities on the near-future buffer performance. To get a better sense of how these sets' capacity really influences the buffer's later behavior, consider the distributions of sqd-*1* and sqd-*0.8*. Under the sqd-*0.8* distribution condition, for instance, *MONPAR* has an impressive succession for all buffer sizes due to the fact that the initial request distribution in spatial dimensions rarely changes –in fact it does with probability of 0.8. The similar succession appears in case of sqd-*1* distribution. However, this success is not as much noticeable as in the former query distribution (sqd-*0.8*), because in sqd-*1* distribution case, others already approach the performance level of *MONPAR*.

In conclusion, *MONPAR* performs better than the other buffer organizations under all query distributions, except the totally random distribution (*uqd*). Even in uqd case, *MONPAR* preserves its preference for small-size and medium-size buffers. As a future work, we plan to analyze statistical approaches like LRU-*K* and their possible integration with structure-aware solutions like *MONPAR* in order to achieve more comprehensive results on buffer management for spatiotemporal indexes.

V. Conclusion

In order to speed up the query executions, disk I/O should be controlled by designing specialized page replacement algorithms. This requirement is becoming more important when we look at the innovations in the area of multi-dimensional databases. In order to design a robust buffer replacement, although we may borrow the ideas from traditional operating system solutions, they are not adequate enough to meet the requirements of database index structures, especially of those indexing multidimensional data. It is the contribution of this paper to propose a specialized buffer for improving the performance of a spatiotemporal index. Essentially, the specialized buffer, namely *MONPAR*, coordinates the pages that should be kept or dropped in the buffer at the cost of analyzing the content of the page.

Although we made a few justifications based on the specialized structural characteristics of MON-tree, it is still possible to manage a fully adaptive buffer on top of current implementation of *MONPAR* In that sense, we are motivated by the possible usage of disproportionate usage of the buffer by different sets. However, we are still not sure that the complexity of such an adaptive organization similar to the one in [2] would not degrade the overall execution time, especially for large buffers, even though it would decrease the number of disk accesses.

References

[1] T.Y. Kahveci, T. Kahveci, A. Singh, "Buffering of Index Structures" in *2000 Proc. SPIE Conf*, Boston.
[2] T. Brinkhoff, "A Robust and Self-tuning Page-Replacement Strategy for Spatial Database Systems" *in 2002 Proc. Conference on Extending Database Technology (EDBT)*, pp. 533-552.
[3] G.M. Sacco, "Index Access with a Finite Buffer" *in 1997 Proc. of the Very Large Data Bases Conf.*, pp. 301-309.
[4] V.T. Almeida, R.H. Güting, "Indexing the trajectories of moving objects in networks" *GeoInformatica vol.9, no.1, 2005*, pp. 33-60.
[5] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Serching" *Proc. ACM SIGMOD, Int'l Conf. Management of Data*, 1984, pp. 47-57.
[6] U. Kalay, O. Kalıpsız, "Probabilistic Point Queries Over Network-based Movements" *Lecture Notes in Computer Science, Springer Verlag,* 2005. *[20th Int'l Conf. ISCIS,, Istanbul*, 2005]
[7] M. Hadjieleftheriou, Spatial Index Library (SaIL), Available: http://u-foria.org/marioh/spatialindex/index.html.