# BRUST: An Efficient Buffer Replacement for Spatial Databases

Jun-Ki Min

School of Internet-Media Engineering
Korea University of Technology and Education
Byeongcheon-myeon, Cheonan, Chungnam,
Republic of Korea, 330-708
jkmin@kut.ac.kr

**Abstract.** This paper presents a novel buffer management technique for spatial database management systems. Much research has been performed on various buffer management techniques in order to reduce disk I/O. However, many of the proposed techniques utilize the temporal locality of access patterns. In spatial database environments, there exists not only the temporal locality, where a recently access object will be accessed again in near future, but also spatial locality, where the objects in the recently accessed regions will be accessed again in the near future. Thus, in this paper, we present a buffer management technique, called BRUST, which utilizes both the temporal locality and spatial locality in spatial database environments.

## 1 Introduction

Spatial database management is an active area of research over the past ten years [1] since the applications using the spatial information such as geographic information systems (GIS), computer aided design (CAD), multimedia systems, satellite image databases, and location based service (LBS), have proliferated. In order to improve the performance of spatial database management systems (SDBMSs), much of the work on SDBMSs has focused on spatial indices [2, 3] and query processing methods [4, 5].

Since data volume is larger than current memory sizes, it is inevitable that disk I/O will be incurred. In order to reduce disk I/O, a buffer is used. Since efficient management of the buffer is closely related to the performance of databases, many researchers have proposed diverse and efficient buffer management techniques.

Well known buffer management techniques utilize temporal locality, where recently accessed data will be accessed in the near future. With SDBMSs, there also exists spatial locality which is the property where objects in recently accessed regions will be accessed in the near future. Therefore, spatial locality should be also considered in buffer management techniques. However, traditional buffer management techniques consider the temporal locality only.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 364–371, 2006.

In this paper, we present a novel buffer management algorithm, called *BRUST* (Buffer Replacement Using Spatial and Temporal locality). In BRUST, using a spatially interesting point (SIP), the buffer management selects a victim which is replaced with a newly access object.

We implemented BRUST and performed an experimental study over various buffer sizes and workloads. The experimental results confirm that BRUST is more efficient than the existing buffer replacement algorithms on SDBMSs environments.

## 2 Related Work

When the buffer is full, buffer management methods find a victim to be replaced with a newly loaded object by analyzing the access pattern using the buffer replacement algorithm. There is a long and rich history of the research performed on buffer management. The core of buffer management techniques is the buffer replacement algorithm. In this section, we present the diverse buffer replacement algorithms.

### 2.1 Traditional Buffer Replacement Algorithm

The most well known buffer replacement algorithm among the various buffer replacement algorithms is LRU [6] (Least Recently Used). The LRU buffer replaces the object which has not been accessed for the longest time (i.e., least recently accessed object). Since the LRU algorithm is based on the simple heuristic rule such that the recently accessed object will be accessed in the near future, the LRU algorithm cannot support diverse data access patterns efficiently.

To overcome this problem, LRU-k [7] was proposed. LRU-k keeps track of the times for the last k references to a object, and the object with the least recent last k-th access will then be replaced. Of particular interest, LRU-2 replaces the object whose penultimate (second to last) access is least recent. LRU-2 improves upon LRU because the second to last access is a better indicator of the inter-arrival time between accesses than the most recent access. LRU-k keeps k-access history for each object and so the process of finding a victim is expensive.

Thus, John and Shasha [8] proposed 2Q which behaves like LRU-2 but is more efficient. 2Q handles the buffer using two separate queues: A1IN and AM. A1IN acts as a first-in-first-out queue and AMQ acts as an LRU queue. When an object not used in the near past is loaded, the object is inserted into A1IN. Otherwise, the object is inserted into AMQ. In 2Q, the history of object replacement is maintained by A1OUT. A1OUT does not contain the object itself. Thus, using the contents of A1OUT, the decision of whether the object was used in the near past or not is made. A disadvantage of 2Q is that the performance of 2Q is determined by the sizes of the queues: A1IN, AM, A1OUT.

The frequency counter and recency history are the major indications of temporal locality. LRFU [9] integrates the two indications. In LRFU, each object $x$ in the buffer has the following value $C(x)$.

$$C(x) = \begin{cases} 1 + 2^{-\lambda} C(x) & \text{if x is referenced at t time} \\ 2^{-\lambda} C(x) & \text{otherwise} \end{cases} \quad (1)$$

In the above formula, $\lambda$ is a tunable parameter. For newly loaded objects, $C(x)$ is 0. When a buffer replacement is required, LRFU selects the object whose $C(x)$ value is smallest as the victim. In LRFU, when $\lambda$ approaches 1, LRFU gives more weight to more recent reference. Thus, the behavior of LRFU is similar to that of LRU. When $\lambda$ is equal to 0, C(x) simply counts the number of accesses. Thus, LRFU acts as LFU. Therefore, the performance of LRFU is determined by $\lambda$.

Practically, it is hard to determine the optimal values for tunable parameters such as $\lambda$ of LRFU and the queue sizes of 2Q. Megiddo and Modha suggested ARC [10] which dynamically changes the behavior of the algorithm with respect to the access pattern. Like 2Q, ARC separates a buffer whose size is $c$ into queues: B1 whose size is $p$ and B2 whose size is $c - p$. B1 and B2 are LRU queues. A newly accessed object is loaded into B1, and an accessed object which is in B1 or B2 is moved into B2. The behavior of ARC is determined by parameter $p$. If a hit occurs on B1, $p$ increases. Otherwise, $p$ decreases. Note that $p$ is not the actual size of B1 but target size of B1. So, the size of B1 may not be equal to $p$. When a buffer replacement occurs, if the size of B1 is greater than $p$, a victim is chosen from B1. Otherwise, a victim is chosen from B2. In this approach, the incremental ratio of $p$ may vary according to the learning rate. In other words, the main goal of eliminating tunable parameters is not accomplished.

Also, LFU-k [11] which is a generalization of LFU has been proposed. And, TNPR [12] which estimates the interval time of accesses for each object was presented. In addition, for buffering an index instead of data, ILRU [13] and GHOST [14] has been suggested.

### 2.2 Buffer Management Techniques for SDBMSs

The buffer replacement algorithms presented in Section 2.1 consider only temporal locality. However, some buffer management techniques for SDBMSs has been proposed.

Papadopoulos and Manolopoulous proposed LRD-Manhattan [15]. In general, a spatial object is represented as an MBR (Minimum Bounded Rectangle) to reduce the computational overhead. Probabilistically, when a point is selected in a unit space, the probability of a large sized object that contains the point is greater than that of a small sized object in the uniform assumption [16]. In other words, large sized spatial objects may be accessed more than small sized spatial objects. LRD-Manhattan computes the average of the access density (i.e., access ratio of each object) and the normalized MBR size of each spatial object. Then, LRD-Manhattan selects a spatial object whose average value is the minimum among all objects in the buffer as a victim.

Recently, ASB [17] which considers the LRU heuristic and the sizes of spatial objects together was proposed. In this technique, a buffer consists of two logically separated buffers, like ARC. The buffer B1 is maintained using the LRU heuristic and the buffer B2 is maintained using the sizes of spatial objects. A

newly accessed object is loaded into B1. When the size of the buffer B1 is insufficient, the least recently used object is moved into the buffer B2. When buffer replacement is required, the object whose MBR size is smallest is selected from B2 as a victim. Also, the sizes of B1 and B2 are incrementally changed with respect to the property of a newly accessed object (see details in [17]).

The techniques presented above utilize a static property (i.e., the size of MBR). Thus, these techniques do not suggest efficient buffer management with respect to dynamic access patterns.

## 3 Behavior of BRUST

In spatial database environments, not only the temporal locality but also spatial locality [18] where spatial queries converge on a specific area of the work space within a certain period exists since user's interest focus on a certain area. In other words, if a certain area is frequently accessed, then the spatial objects near that area have high probability of being accessed.

When considering only temporal locality, a spatial object in the frequently accessed area may be a victim. In this case, the efficiency of the buffer is degraded. Also, if we only consider spatial locality, some access patterns such as random access and liner scan are not efficiently supported.

Therefore, in contrast to the previous techniques, the proposed buffer management technique, BRUST, considers temporal locality and spatial locality together. The basic heuristic rule of BRUST is that a spatial object near the frequently used area will be used in the near future.

First, in order to utilize the spatial locality, we estimate a spatially interesting point (SIP) with respect to the spatial locations of accessed objects. The initial location of a SIP is the center point of workspace. Let the currently estimated SIP be $< x_{sip}, y_{sip} >$ and a spatial object whose center point p $< x_p, y_p >$ is be accessed. Then a SIP should be modified in order the reflect the change of user's interest. The following formula is for the estimation of a new SIP.

$$\begin{aligned} x_{sip} &= x_{sip} + w_x \cdot (x_{sip} - x_p) \\ y_{sip} &= y_{sip} + w_y \cdot (y_{sip} - y_p) \end{aligned} \tag{2}$$

During the estimating phase of a SIP, we consider outliers since, in general, a user interesting location is gradually changed. Thus, as presented in equation (2), the weight factors $w_x$ *and* $w_y$ are multiplied. As distance between the current SIP and a newly accessed object increases, the probability that the location of a newly accessed object is a user interesting location dramatically decreases. Thus, we use a simple decrease function $(1/e)^{\lambda}$. As described in equation (3), $w_x$ decreases as the distance between the current SIP and a newly accessed object increases.

$$\begin{aligned} &w_x = (\tfrac{1}{e})^{|x_{sip} - x_p| / Domain_x} \\ &\text{where } e \text{ is the base of natural logarithm,} \\ &\quad \text{and } Domain_x \text{ is the length of workspace on x-coordinate} \end{aligned} \tag{3}$$

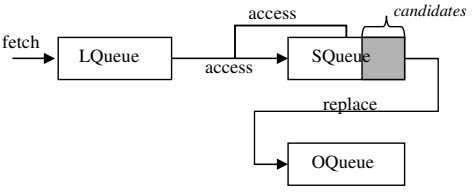


**Fig. 1.** The behavior of BRUST

The behavior of BRUST is described in Figure 1. As depicted in Figure 1, there are three queues in BRUST. Among the queues, LQueue contains the objects which have been accessed once recently. SQueue contains the objects which have been accessed at least twice recently. Conceptually, LQueue maintains the recently accessed objects and SQueue maintains the frequently accessed objects. The sizes of LQueue and SQueue are adaptively changed like ARC. OQueue does not contain objects like AOUT queue in 2Q but keeps the replacement history. The size of OQueue is 30% of the size of SQueue.

If a buffer replacement is required, BRUST finds the farthest object from a SIP in SQueue and makes the farthest object a victim. To find out the farthest object in SQueue, all objects in SQueue should be evaluated. It degrades the efficiency of the buffer management. Thus, in BRUST, a victim is selected from the portion of SQueue, called *candidates*. The gray box in Figure 1 represents *candidates*.

Now we have an important question which concerns the size of *candidates*. The basic idea of the problem is that the size of *candidates* is changed with respect to the access pattern.

Note that BRUST selects a object in *candidates* as a victim. The case that a replaced object in the near past is re-referenced means the influence of temporal locality is larger than that of spatial locality. Recall that OQueue exists in BRUST in order to keep the replace history. Thus, if a newly access object is found in OQueue, BRUST reduces the size of *candidates* in order to reduce the effect of spatial locality. Otherwise, BRUST increases the size of *candidates*.

## 4 Experiments

In this section, we show the effectiveness of BRUST compared with the diverse buffer management techniques: LRU, 2Q, ARC and ASB. As mentioned earlier, LRU, 2Q and ARC consider only temporal locality. And ASB consider temporal locality and the property of s spatial object (i.e., the size of MBR). We evaluated the hit ratio of BRUST using the real-life and synthetic data over various sized buffers. However, due to the space limitation, we show only the experimental result of the real-life data when buffer sizes are 5%, 10% and 20% of total size of objects. The real-life data in our experiments were extracted from TIGER/Line data of US Bureau of the Census [19]. We used the road segment data of Kings county of the California State. The number of spatial objects is 21,853 and the size of work space is 840,681×700,366.

To evaluate the effectiveness of BRUST, we made 3 kinds of access pattern. First, we made an uniform access pattern (termed *Uniform*) where all spatial objects have same access probability. In order to measure the effect of temporal locality, we made a temporally skewed access pattern (termed *Time Skew*) using Zipf distribution. In temporally skewed access pattern, 80% of the references accesses 20% of spatial objects. Finally, we made a spatially skewed access pattern (termed *Spatial Skew*). In this pattern, 90% of the references access the spatial objects which are in a 10% sized region of the work space. Thus, in the spatially skewed access pattern, temporal locality and spatial locality appear intermixedly. In each access pattern, 1,000,000 accesses of spatial objects occur.

The following figures presents the experimental results over diverse sized buffers.

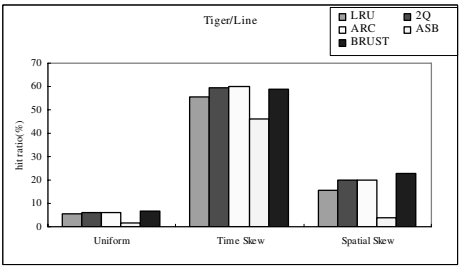


**Fig. 2.** The results on the 5% sized buffer

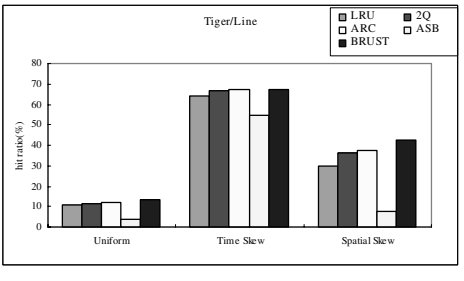


**Fig. 3.** The results on the 10% sized buffer

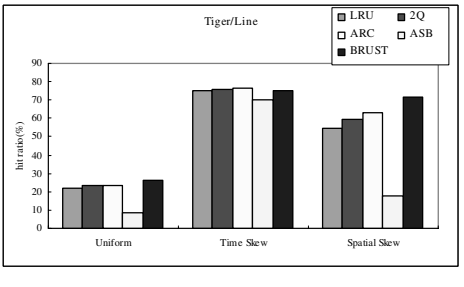


**Fig. 4.** The results on the 20% sized buffer

As shown in Figure 2, Figure 3, and Figure 4, BRUST shows the best hit ratio over most of all cases. Of particular, BRUST outperforms LRU, 2Q, ARC, and ASB on the *Time Skew* workload over diverse sized buffers even though BRUST considers temporal locality and spatial locality together.

ASB shows the worst performance over most of cases since a victim is selected with respect to a static property (i.e., MBR size). LRU does not show the most efficient performance but not show the worst performance over all cases. 2Q and ARC show good performance on the *Time Skew* workload since 2Q and ARC is devised for the temporally skewed accesses.

BRUST shows the better performance than ARC on the *Time Skew* workload with the real-life data. Recall that BRUST dynamically changes the sizes of SQueue and OQueue like ARC. In addition, BRUST adaptively changes the sizes of *candidates* with respect to the effect of a SIP. In the real-life data set, locations of spatial objects are clustered. Thus, spatial locality occurs in the Time Skew workload although it is not intended. Therefore, BRUST is superior to ARC.

Consequently, BRUST is shown to provide best performance over diverse access patterns with various sized buffers. Of particular, BRUST is superior to the other buffer replacement algorithm in spatial database environments (i.e., *Spatial Skew* workload).

## 5 Conclusion

In this paper, we present a novel buffer management technique, called BRUST (Buffer Replacement Using Spatial and Temporal locality) which consider spatial and temporal locality together. BRUST handles a buffer using two queues: LQueue and SQueue. And, OQueue is used to keep the object replacement history. The sizes of queues are dynamically changed. In BRUST, a spatial interesting point (SIP) is estimated in order to utilize the spatial locality. A victim, which is the farthest object from a SIP, is selected from a portion of SQueue, called *candidates*. The size of *candidate* is adaptively changed with respect to the influence of the spatial and temporal locality.

To show the effectiveness of BRUST, we conducted an extensive experimental study with the synthetic and real-life data. The experimental results demonstrate that BRUST is superior to existing buffer management techniques in spatial databases environments.

**Acknowledgement.** This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment) (IITA-2005-C1090-0502-0016)

## References

1. Guting, R.H.: An introduction to spatial database systems. VLDB **3** (1994) 357–399
2. Guttman, A.: The R-tree: A Dynamic index structure for spatial searching. In: Proceedings of ACM SIGMOD Conference. (1984) 47–57

3. Brinkhoff, T., Kriegel, H., Scheneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proceedings of ACM SIGMOD Conference. (1990) 322–331
4. Min, J.K., Park, H.H., Chung, C.W.: Multi-way spatial join selectivity for the ring join graph. Information and Software Technology **47** (2005) 785–795
5. Papadias, D., Mamoulis, N., Theodoridis, Y.: Processing and Optimization of Multiway Spatial Join Using R-Tree. In: Proceedings of ACM PODS. (1999) 44–55
6. Effelsberg, W.: Principles of Database buffer Management. ACM TODS **9** (1984) 560–595
7. O'Neil, E.J., Neil, P.E.O., Weikum, G.: The LRU-K Page Replacement algorithm for database disk buffering. In: Proceedings of ACM SIGMOD Conference. (1993) 297–306
8. Johnson, T., Shasha, D.: 2Q: a Low Overhead High Performance Buffer Management Replacement Algorithm. In: Proceedings of VLDB Conference. (1994) 439–450
9. D. Lee, J.C., Kim, J.H., Noh, S.H., Min, S.L., Cho, Y., Kim, C.S.: LRFU: A Spectrum of Policies that subsumes the Least Recently Used and Least Frequently Used Policies. IEEE Tans. Computers **50** (2001) 1352–1360
10. Megiddo, N., Modha, D.S.: ARC: A Self-tuning, Low Overhead Replacement Cache. In: Proceedings of USENIX FAST Conference. (2003)
11. Sokolinsky, L.B.: LFU-K: An Effective Buffer Management Replacement Algorithm. In: Proceedings of DASFAA. (2004) 670–681
12. Juurlink, B.: Approximating the Optimal Replacement Algorithm. In: ACM CF Conference. (2004)
13. Sacco, G.M.: Index Access with a Finite Buffer. In: Proceedings of VLDB Conference. (1987)
14. Goh, C.H., Ooi, B.C., Sim, D., Tan, K.: GHOST: Fine Granularity Buffering of Index. In: Proceedings of VLDB Conference. (1999)
15. Papadopoulos, A., Manolopoulos, Y.: Global Page Replacement in Spatial Databases. In: Proceedings of DEXA. (1996)
16. Kamel, I., Faloutsos, C.: On Packing R-Trees. In: Proceedings of CIKM. (1993) 490–499
17. Brinkhoff, T.: A Robust and Self-tuning Page Replacement Strategy for Spatial Database Systems. In: Proceedings of DEXA. (2002) 533–552
18. Ki-Joune, L., Robert, L.: The Spatial Locality and a Spatial Indexing Method by Dynamic Clustering in Hypermap System. In: Proceedings of SSD. (1990) 207–223
19. Bureau, U.C.: UA Census 2000 TIGER/Line Files. (http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html)