On the Cost of Monitoring and Reorganization of Object Bases for Clustering

Carsten A. Gerlhof[*] *Alfons Kemper*[*] *Guido Moerkotte*[†]

[*] Universität Passau
Lehrstuhl für Informatik
94030 Passau, Germany
[*gerlhof* | *kemper*]@*db.fmi.uni-passau.de*

[†] Universität Mannheim
Lehrstuhl für Praktische Informatik III
68131 Mannheim, Germany
moer@pi3.informatik.uni-mannheim.de

Abstract

Clustering is one of the most effective means to enhance the performance of object base applications. Consequently, many proposals exist for algorithms computing good object placements depending on the application profile. However, in an effective object base reorganization tool the clustering algorithm is only one constituent. In this paper, we report on our object base reorganization tool that covers all stages of reorganizing the objects: the application profile is determined by a monitoring tool, the object placement is computed from the monitored access statistics utilizing a variety of clustering algorithms and, finally, the reorganization tool restructures the object base accordingly. The costs as well as the effectiveness of these tools is quantitatively evaluated on the basis of the OO1-benchmark.

1 Introduction

Ever since the "early days" of database management systems, clustering has proven to be one of the most effective performance enhancement techniques. Therefore, many clustering algorithms have been proposed. These algorithms can roughly be classified into sequence-based clustering algorithms [1, 2, 4, 9, 10, 12] and partition-based clustering algorithms [8, 13]. For an overview see [11]. Clustering algorithms have to rely on information about the application's access behavior or an abstraction thereof. This information must either be guessed by a database administrator or is gathered during a monitoring phase. Based on this information the clustering strategy computes a new—hopefully better—physical organization (i.e., object placement scheme) of the object base. According to this object placement, the object base is reorganized.

While many clustering algorithms have been proposed and empirically evaluated, there appears to be no published work on the cost of monitoring and reorganization. Since different clustering algorithms exploit different kinds of information, different monitoring costs are induced. Even worse, so far no techniques for monitoring object base access patterns have been proposed. Likewise, we are not aware of a detailed analysis of object base reorganization methods. In order to fill this gap, the paper introduces and evaluates both, monitoring as well as reorganization implementations. The induced overhead as well as the resulting performance enhancement due to clustering are evaluated with the MERLIN storage system. Rather than relying on simulations (as is common practice) we chose to analyse real experiments.

All three different services—monitoring, clustering algorithms, and database reorganization (MCR abbreviated)—constituting the three phases of an object base restructuring are integrated into the MCR tool box. The interrelationships between the different tools within the tool box are described in Section 2. Then, in Section 3, we briefly review some clustering algorithms. It is important to note, that our goal is not to introduce a new clustering strategy but to give solutions for their practical application as well as an evaluation of their performance gain in relation to their overhead. Hence, for a more complete and detailed overview of clustering algorithms, the reader should refer to [11]. Section 4 discusses various monitoring techniques for the different kinds of information the clustering algorithms rely upon. Section 5 discusses database reorganization and Section 6 evaluates the performance improvement achieved by the reorganization. Finally, Section 7 concludes the paper.

2 The Tool Kit MCR

Fig. 1 shows an overview of the components constituting the tool kit and indicates the dataflow between the different tools. On the left-hand side, an application running on the object base management system is indicated. The example application we used for benchmarking as well as the configuration of the underlying object base will be described in Section 6. While the application is running, the monitoring tool collects information about its access behavior. The information gathered is described in the Section 4. The clustering tool is parameterized by the desired algorithm. Therefore, we analyse a variety of clustering algorithms. The input of this tool is the monitored profile. Based on this information, the clustering tool computes an object placement map which maps each object to a particular page. More specifically, for each old physical object identifier a new physical object identifier is generated. Note that this necessitates the adjustment of references—under the premise that they are

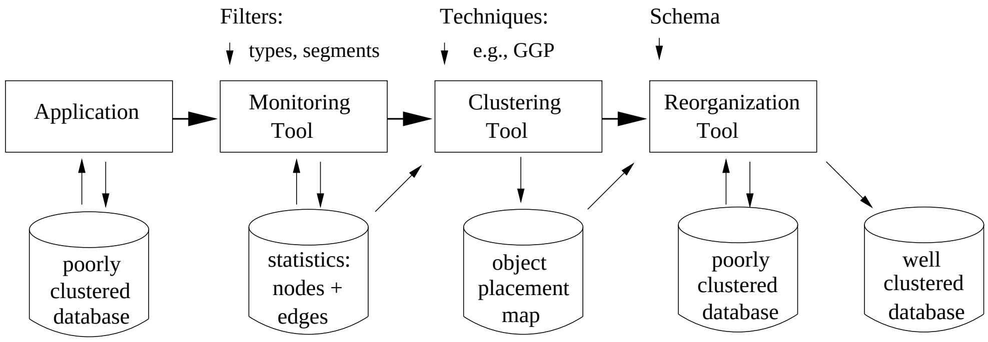

Figure 1: Overview.

implemented as physical object identifiers. In a last step, the reorganization tool uses the map as well as the old object base in order to generate the new reorganized object base.

3 Clustering Techniques

Modeling the Application Typically, the input to the clustering algorithms is not the monitored access trace. Rather, an abstraction of the applications' behavior—the clustering graph (CG)—is used. It is defined as follows: Each object constitutes a node in the CG. For the edges, two possibilities exist: directed edges and undirected edges. Whether they are directed or undirected depends on the clustering algorithm used. Additionally, with each node and each edge there is an associated weight. The weight of a node constitutes the number of times it was accessed during the monitoring phase. The weights of the edges reflect the number of consecutive accesses of the two objects. Not every clustering algorithm makes use of all of this information but since we want to be able to incorporate a variety of clustering algorithms invented, we provide all these monitoring techniques.

Mapping Objects onto Pages In principle, the clustering tool could utilize any of the known clustering algorithms. But so far, we have not implemented all of them. The task of the clustering tool is to partition the nodes of the CG into partitions with the page size as an upper limit on the sum of the objects' sizes in each partition. Thus, the resulting partitions fit on a single page each. Reflecting our experience with the evaluation of clustering algorithms [7, 8], we decided to implement the following algorithms:

1. Probability Ranking (PRP) [15]
2. Best First Traversal (BSTF) [10]
3. Greedy Graph Partitioning (GGP) [7, 8]

BSTF is the only clustering algorithm in need of some additional parameters: UDE specifies that undirected edges are to be used, NPNT demands to start a new traversal when the current page is filled and a new one is needed, and NTNP indicates that for each new traversal a new page is used. These parameters are common and crucial for all traversal-based clustering algorithms [8]. In this paper, NPNT and NTNP were tuned such that the best clustering result is achieved. Additionally, BSTF and GGP have a subsequent clean up phase where (less than) half-filled pages are merged.

4 Monitoring

Two kinds of information have to be gathered during the monitoring phase:

1. *reference counts*: the number of times an object is accessed, and
2. *link counts*: the number of times two object identifiers id_i and id_j occur consecutively within the trace.

Our object manager MERLIN is able to report this information to the monitoring tool. The task of the latter then is to accumulate and store this information.

Recording *reference counts* is quite simple. The object identifiers are simply inserted into an index. They are used as keys. Additionally, we use two data fields: (1) the *weight* which accumulates the number of times the object with the corresponding object identifier is referenced and (2) the *size* of the object with the corresponding identifier.

The procedure for recording the *link counts* depends on the fact whether the edges in the clustering graph have to be directed or not. An *undirected edge* (id_1, id_2) is stored such that $id_1 < id_2$ (lexicographic order) holds. This allows to locate an edge efficiently in an index regardless in which direction the link is traversed, because the smaller object identifier is used as a key. For a *directed edge*, we use the first object identifier as the key. In case of BSTF we have two choices: We could store only one version, but with a counter for each direction. Then, the second index must be explicitly built after monitoring—this approach is similar to collecting undirected edges for BSTF. We decided to store both versions separately in order to cover both cases. The data used for link counts consists of (1) the weight of the link and (2) the sizes of the participating objects. In case of BSTF, we needed an additional data field to be able to mark already visited objects in the subsequent mapping phase. Note that this is not necessary for PRP and GGP.

In case both parameters—node weights and edge weights—are needed, we omit the index for nodes during monitoring. Instead, we compute the node weight as the sum of all edge weights pointing to the object. The computed node weights are then stored in the node index in a separate phase. The advantage of this approach is that this accumulation can be carried out offline and, therefore, an additional slow down of the monitoring phase is avoided. In all cases, we use a B^+-Tree as an index structure. We also use the B^+-Trees to sort nodes and edges according to their weights. Some additional tuning of the monitoring and the subsequent phases is possible. We facilitate the monitoring, reclustering and reorganization of parts of the database. These parts are described either on a logical level by specifying a list of types or on a physical level by specifying a list of segments.

5 Reorganization

After running the clustering algorithms, the reorganization phase takes place. The input to the reorganization tool is the *object placement map*. It consists of a mapping of old physical object identifiers to new physical object identifiers. Since we use physical object identifiers, the new placement of the objects can be derived from their identifiers. If we used logical object identifiers, the placement map would have to contain the new address explicitly.

Given the object placement map as input, the reorganization tool has to perform three tasks: (1) placing the objects according to the placement map, (2) adjusting the references contained within the objects, and (3) adjusting object identifiers contained in indexes. The latter two tasks are necessary since the old physical object identifiers are no longer valid after the objects have been moved. This task could be omitted, if instead of physical object identifiers logical object identifiers were used. Although the additional overhead of logical object identifiers for object lookup is tolerable [6], most commercial and experimental systems still use physical object identifiers. This is the reason we used them in this experiment. Considering the additional costs induced by adjusting the references contained in objects and reorganizing the indexes, logical object identifiers are clearly superior in this context. (See also next section.)

An alternative to our approach would be to use a bulk loading facility as described in, e.g., [14].

6 Evaluation

We used the OO1 benchmark [3] with a slight modification. In the original OO1 benchmark, the database contains *part* and *connection* objects where each part has exactly three connections to other parts. We varied only the selection mechanism for those part objects that participate in a connection: The first connection of each part was used to link all part objects to a ring; the other two connections are chosen randomly.

The operation is a depth-first traversal (forward direction) to level seven as described in [3]. This operation was executed on the database several times, whereby on each invocation a start object (part) is chosen randomly. This facilitates for (1) the adjustment of locality by increasing the number of repetitions and (2) modeling applications with different trace lengths.

We used two databases: a small database with 23,000 part objects and 69,000 connection objects, and a larger one with 120,000 part objects and 360,000 connection objects. The part objects varied in size because in our object system MERLIN small collections are inlined, i.e., directly stored into the objects. The buffer size was 20% of the database size. The page size was 4KB. Table 1 summarizes the statistics made during a varying number of traversals. For each number of repetitions, the number of references and the number of nodes and edges in the clustering graph are reported. Since the number of directed edges and undirected edges did not differ significantly, the latter is omitted. In addition, the total size (in pages) of the clustering graph is given under different monitoring techniques.

The next sections evaluate the costs of monitoring, applying a clustering algorithm, and reorganization. Further, we report on the gain of reclustering under different strategies.

6.1 Monitoring

Fig. 2 shows the performance of the OO1 forward traversal in terms of page faults with and without monitoring. The best case exhibits only a 34% increase in running time (at 500 repetitions). As expected, collecting directed edges was much more expensive than collecting undirected edges, but—to our own surprise—collecting nodes for PRP induced significantly more page faults than collecting undirected edges.

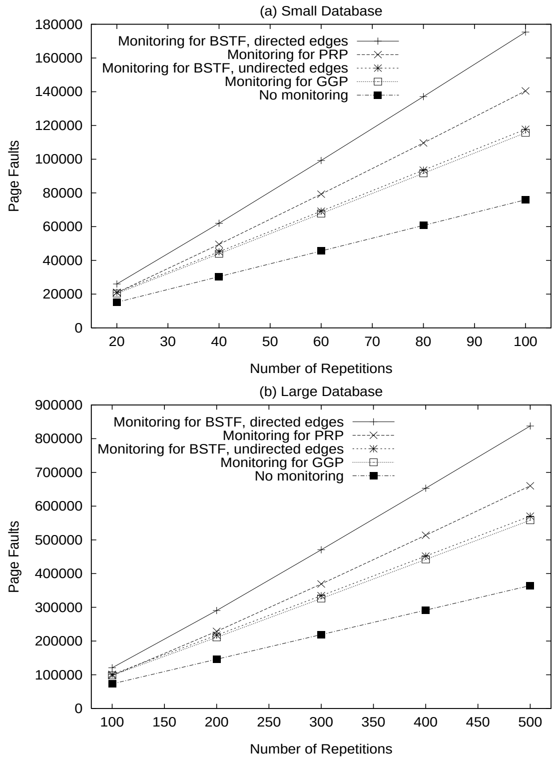

Figure 2: The Cost of Monitoring.

#repet.	#references	#nodes	#edges	PRP-index	GGP-index	BSTF-index
100	655 900	155 655	186 616	2151	3057	3578
200	1 311 800	242 558	324 152	3347	5226	6213
300	1 967 700	297 717	428 624	4131	6948	8283
400	2 623 600	335 806	511 634	4671	8243	9889
500	3 279 500	363 183	578 472	5058	9400	11118

Table 1: Database Statistics.

To explain this effect, consider the following example: From the object trace

$$\ldots, id_{20}, id_{10}, id_{40}, id_{30}, id_{60}, id_{50}, id_{80}, \ldots$$

we would derive the following sequence of undirected edges by using the technique described in Section 4:

$$\ldots, (id_{10}, id_{20}), (id_{10}, id_{40}), (id_{30}, id_{40}), (id_{30}, id_{60}), \ldots$$

A sample storage structure is depicted in Figure 3. Now, assume that all directory pages of the B$^+$-Tree and one leaf page can be cached. On updating the statistics, it is very likely that almost every second edge has a hit in the cache, because the edges are clustered according to the first node. In contrast, an update of a node's weight suffers from poor locality, because on a random walk through the database it is not likely that the OIDs of two objects which are subsequently accessed are also adjacent in the leaf page of the B$^+$-Tree.

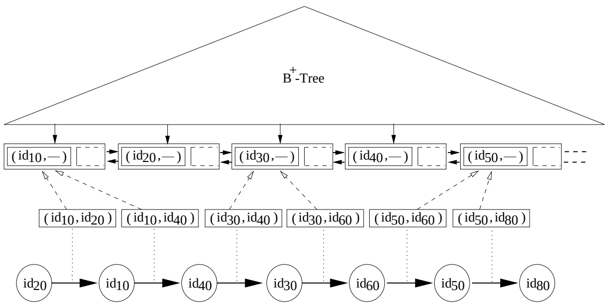

Figure 3: Sample Storage Structure.

6.2 Compiling of the Statistics

Fig. 4 shows the number of page faults encountered during the compilation of the collected statistics. Sorting nodes according to the reference counts produce only negligible costs. This also holds for sorting undirected edges according to the transition frequencies. As expected, sorting nodes is a little bit faster then sorting edges. However, in case of BSTF sorting undirected edges incurred costs an order of magnitude higher than every other sorting. The reason is that the nodes need to be sorted and, for the sake of symmetry, additional edges must be inserted since BSTF needs direct access to both nodes of an edge. This outweights the advantage of collecting undirected edges for BSTF. Thus, regardless which version we use, BSTF either has to pay the price in the monitoring phase (due to directed edges) or it has to pay the price in the sorting phase (due to undirected edges).

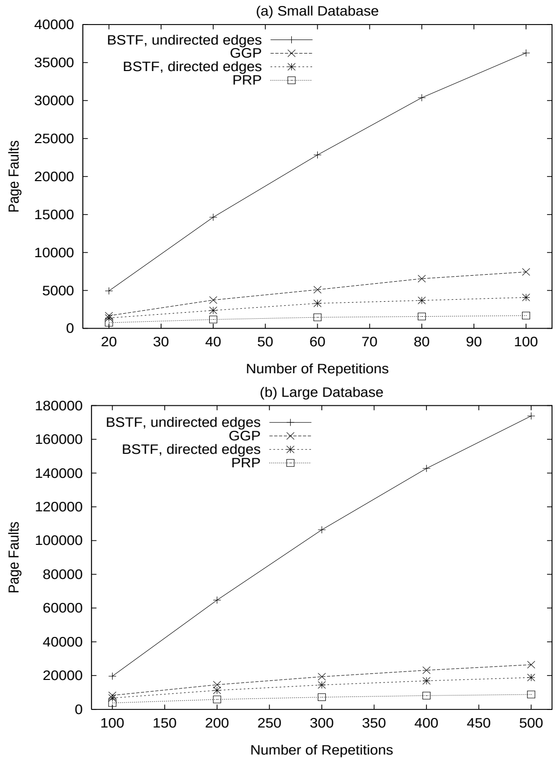

Figure 4: The Cost of Statistics Compilation.

6.3 Computation of the New Clusters

Fig. 5 shows the page faults incurred by the mapping techniques. PRP is very efficient, because it only needs to scan through the index in order to build the placement map. GGP ranks second, its performance is still acceptable. Similarly to PRP, GGP scans the edge list by following the leaf pages of the B$^+$-Tree. However, GGP needs an additional index to record the current partition of an object. BSTF walks through the object net and, therefore, causes many more page faults. This is even more severe for BSTF with undirected edges. Here, BSTF suffers significantly from the increased storage overhead since its index structure contains twice as many edges as the version with directed edges. Note that BSTF marks objects during its traversal; thus, write operations induce higher cost for BSTF than for GGP.

6.4 Moving Objects and Updating References

In this paper, we used only *physical* OIDs; thus, all object references in the database needed to be updated. Additionally, index structures had to be patched. We had

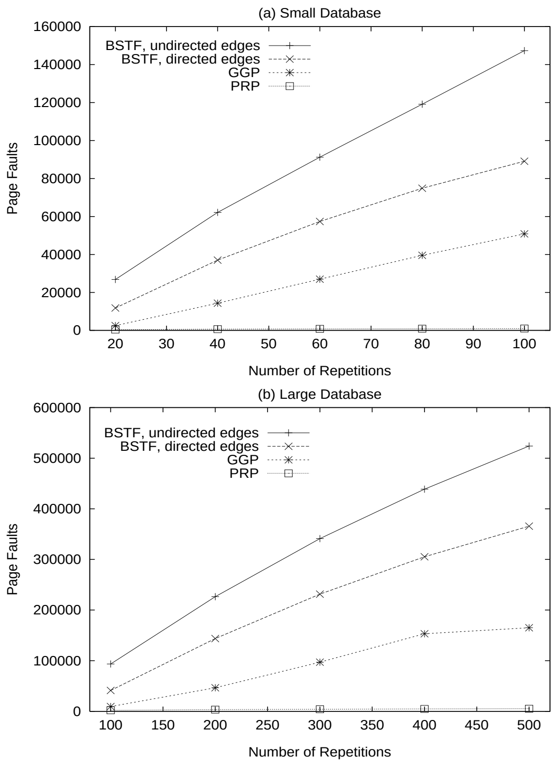

Figure 5: The Cost of Clustering.

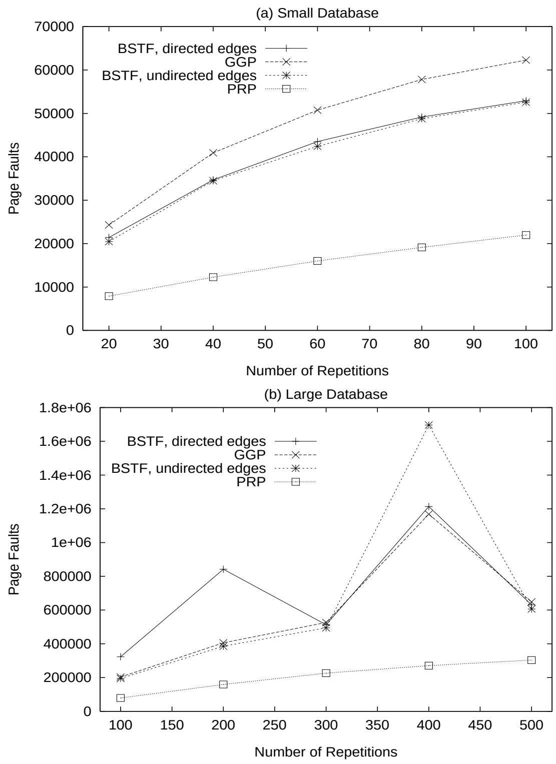

Figure 6: The Cost of Reorganization.

three index structures: two for the type extensions and one for the part numbers in order to efficiently select a start object without scanning the type extension. The induced costs are given in Fig. 6. In our experiments, the costs for moving the objects amounted to only 25% of the total costs. Hence, we can conclude that the costs of a database reorganization under use of physical OIDs strongly depends on the number of existing index structures. This clearly advocates the use of logical OIDs for which very efficient implementation techniques exist [6].

6.5 Clustering Quality

Finally, we address the important question: was clustering worth the effort? Fig. 7 gives the answer. In this diagram, we also included the performance of the OO1 traversal on a database which initially contained *forwards* because on insertion of new connections some objects' size increased and, thereby, were moved from their home page. This gives an impression of the speed-up one can expect in the best case and in the worst case. Fig. 7 shows that in terms of clustering quality GGP was the winner. Compared to the database with forwards there was an improvement of 55%, and compared to the database without forwards there was an improvement of 37%. These experiments demonstrated that BSTF with undirected edges performed much better (ranked second) than BSTF using directed edges (ranked third). This is because in case of undirected edges BSTF can take into account more possibilities than in case of "handicapped" edges. PRP was the clear loser. At many repetitions, PRP clustering even performed worse than the original database. This was mainly due to the fact that PRP was particularly penalized by this specific kind of profile. However, PRP has its justification in applications with an IR-profile [5]. In summary, these numbers confirm the results obtained in previous work [8].

6.6 Profitability of Clustering

Looking at the total overhead—monitoring, clustering, and reorganization—the question arises whether these costs amortize. To answer the question, we divided the total reorganization costs by the gain due to the reorganization. This factor then gives the number of times an application must be executed in order for the reorganization to pay off. Table 2 summarizes these factors for both database sizes and for databases with and without forwards.

Small DB	Number of repetitions				
	20	40	60	80	100
no forwards	3.2	4.5	5.2	5.5	5.6
with forwards	2.0	2.6	2.7	2.8	2.7

Large DB	Number of repetitions				
	100	200	300	400	500
no forwards	4.6	6.6	7.4	12.6	7.5
with forwards	3.0	3.9	4.0	6.4	3.7

Table 2: Profitability of Clustering.

Let us look at the numbers for databases with forwards first. For the small database reorganization already pays off after 2–3 runs of the application. Also, for the large

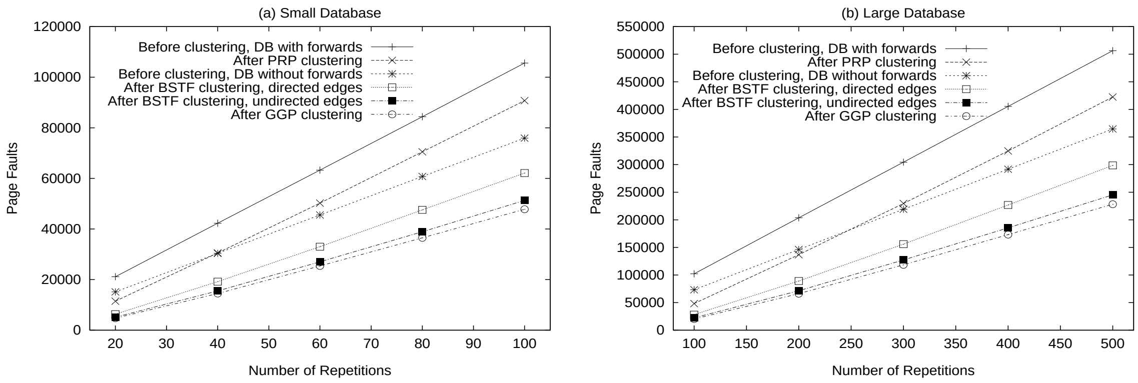

Figure 7: Performance of Clustering Techniques.

database where reorganization pays off after 3–7 runs, it can strongly be recommended to reorganize frequently — if the application or the database profile changes. For databases without forwards the amortization factors are 4–6 and 5–13 for the small and the large databases, respectively. Although slightly higher, these factors still indicate that reorganization can very well pay off. This is especially true when considering the fact that the MCR toolbox is capable of partial reorganizations.

7 Conclusion

Within the context of object-oriented databases, this paper for the first time treats the entire task of monitoring, clustering, and reorganization (MCR) of the database. Besides giving the implementation techniques involved in implementing such a system, performance numbers for the three constituents as well as on the gain of reclustering are given. These numbers were not drawn from simulations but from measurements of a real storage management system.

The most essential finding is that the cost of reorganization is amortized after relatively few repeated runs of the monitored applications. This indicates that an implementation of a reorganization toolkit as MCR is a worthwhile means to enhance the performance of object-oriented database management systems.

Acknowledgments Stefan Augustin implemented the MCR tool kit on top of MERLIN.

This work was supported by the German Research Council DFG (contracts Ke 401/6-1,2 and Ke 401/7-1).

References

[1] J. Banerjee, W. Kim, S. J. Kim, and J. F. Garza. Clustering a DAG for CAD databases. *IEEE Trans. on Software Eng.*, 14(11):1684–1699, November 1988.

[2] V. Benzaken and C. Delobel. Enhancing performance in a persistent object store: Clustering strategies in O_2. In A. Dearle, G. Shaw, and S. Zdonik, editors, *Implementing Persistent Object Bases*, pages 403–412, Martha's Vineyard, September 1990. Morgan Kaufmann.

[3] R. Cattel and J. Skeen. Object operations benchmark. *ACM Trans. on Database Systems*, 17(1):1–31, 1992.

[4] J. R. Cheng and A. R. Hurson. Effective clustering of complex objects in object-oriented databases. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 22–32, Denver, CO, May 1991.

[5] E. Coffman and P. Denning. *Operating Systems Theory.* Prentice Hall, Englewood Cliffs, NJ, USA, 1973.

[6] A. Eickler, C. Gerlhof, and D. Kossmann. A performance evaluation of OID mapping techniques. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 18–29, Zurich, 1995.

[7] C. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte. Clustering in object bases. Technical Report 6/92, University of Karlsruhe, D-76050 Karlsruhe, Germany, 1992.

[8] C. A. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte. Partition-based clustering in object bases: From theory to practice. In *Proc. of the Intl. Conf. on Foundations of Data Organization and Algorithms (FODO)*, volume 730 of LNCS, pages 301–316, Chicago, IL, October 1993. Springer-Verlag. (an extended version is available as RWTH Aachen Technical Report 92-34, RWTH Aachen, D-52056 Aachen, December 1992).

[9] M. F. Hornick and S. B. Zdonick. A shared, segmented memory system for an object-oriented database. *ACM Trans. on Office Inf. Syst.*, 5(1):70–95, January 1987.

[10] S. E. Hudson and R. King. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. *ACM Trans. on Database Systems*, 14(3):291–321, September 1989.

[11] A. Kemper and G. Moerkotte. *Object-Oriented Database Management: Applications in Engineering and Computer Science.* Prentice Hall, Englewood Cliffs, NJ, USA, 1994.

[12] J. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Transactions on Computer Systems*, 2(2):155–180, May 1984.

[13] M. Tsangaris and J. F. Naughton. On the performance of object clustering techniques. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 144–153, San Diego, USA, June 1992.

[14] J. Wiener and J. Naughton. OODB bulk loading revisited: The partitioned-list approach. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 30–41, Zurich, 1995.

[15] P. C. Yue and C. K. Wong. On the optimality of the probability ranking scheme in storage applications. *Journal of the ACM*, 20(4):624–633, October 1973.