

Cache Conscious Clustering C3

Zhen He and Alonso Marquez

Department of Computer Science, The Australian National University
Canberra, ACT 0200, Australia
{zhen, alonso}@cs.anu.edu.au

Abstract. Despite over 10 years of research into OODBMS design, performance remains as one of the major problems. I/O reduction has proven to be one of the most effective ways of enhancing performance. Two techniques of improving I/O performance of OODBMS are *clustering* and *buffer replacement*. *Clustering* places objects that are likely to be accessed together onto the same disk page and thus reduces I/O. *Buffer replacement* involves the selection of a page to be evicted, when the buffer is full. The page evicted ideally should be the page needed furthest in the future. Selection of the correct page for eviction results in a reduction in the total I/O generated by the system. These two techniques effect the likelihood of a requested object being memory resident in an interdependent way. This fact has been acknowledged in the existing literature. However despite this acknowledgement no existing work investigates this interdependency. This paper makes the first investigation into this interdependency by exploring the effects of ten different buffer replacement algorithms on the performance of two different clustering algorithms. We developed a new family of clustering algorithms that incorporate cache behavior when performing clustering. We term this new family of clustering algorithms, *cache conscious clustering (C3)*. A particular C3 algorithm (GPC3) was tested against a well known and highly competitive clustering algorithm GGP, and the results show GPC3 outperforms GGP by upto 40% for popular buffer replacement algorithms such as LRU and CLOCK. These results show for the first time clustering should be approached from a cache conscious point of view.

1 Introduction

The current rate of performance improvement for CPUs is much higher than that for memory or disk I/O. CPU performance doubles every 18 months while disk I/O improves at only 5-8 % per year. In addition, cheap disks mean object databases will become bigger as database designers realise that more data can be stored [9]. A consequence of these facts is that disk I/O is likely to be a bottleneck in an increasing number of database applications. It should also be noted, memory is also becoming a more prevalent source of bottleneck on modern DBMS [1]. However their study was conducted on relational DBMS. We believe for object-oriented DBMS where navigation is common, I/O may be a more common source of bottleneck.

The question now is what is the best way of reducing the effect of I/O in an OODBMS? We believe the answer lies in taking advantage of the synergy that exists between the optimisation techniques, *clustering*, and *buffer replacement*. *Clustering* is the arrangement of objects into pages so that objects accessed close to each other temporally are placed into the same page. This in turn reduces the total I/O generated. *Buffer replacement* involves the selection of a page to be evicted, when the buffer is full. The page evicted ideally should be the page needed furthest in the future. Selection of the correct page for eviction results in a reduction in the total I/O generated by the system. This paper will show that synergy does indeed exist between the two techniques. In addition, exploitation of that synergy results in improvements in performance. In one of the most fundamental papers on clustering [14], Tsangaris and Naughton stated that, “A clustering strategy will depend much on the cache management policy.” However despite this statement they chose not to explore this dependency, stating as the reason, “having to deal with the cache policy makes clustering an even more difficult problem.” [14]. The paper then goes on to state the aim of clustering should therefore be to “enhance locality of the client reference stream, using a metric independent of the cache management policy.” Finally the paper proposes the *graph partitioning* family of algorithms as the best way to cluster objects for enhanced locality.

This paper proposes a new family of clustering algorithms termed cache conscious clustering algorithms (C3). Cache conscious clustering algorithms exploit the synergy between clustering and buffer replacement. In order to evaluate the performance of C3, we conducted experiments involving GPC3 and Greedy Graph Partitioning (GGP) [7]. GPC3 is a concrete example of a C3 algorithm. GGP is a member of the graph partitioning family of clustering algorithms. Up to now graph partitioning has been shown to give the best quality ¹ of clustering [13, 7]. GPC3 was found to outperform GGP in a variety of situations.

There are three main contributions made by this paper. The first contribution is restating the clustering problem to incorporate caching. The second, is proposal of a new family of clustering algorithms that incorporate caching. We then give a concrete example of a new clustering algorithm that falls within this new family of clustering algorithms. This new clustering algorithm is evaluated in a simulation study. It should be noted this algorithm is the first static clustering ² algorithm that exploits the dependency between clustering and cache management. The third contribution is a simulation study of the dependency between clustering and cache management.

¹ By quality of clustering we mean the ability to minimise I/O in perfect conditions. Perfect conditions occur when the pattern of object access used for training and evaluation are exactly the same.

² By static clustering, we mean a clustering algorithm that rearranges the object base when the database is off-line.

2 Related Work

There has been a number of existing studies on static clustering algorithms [13, 7]. The simplest static clustering algorithm is the Probability Ranking Principle (PRP) algorithm [13]. PRP just involves placing the objects in the object graph in decreasing heat (where ‘heat’ is simply a measure of access frequency). This simple approach groups objects of similar heat into the same page. When the buffer is large and the working set of the database completely fits into memory this algorithm provides the optimal solution.

Graph partitioning clustering algorithms consider the object placement problem as a graph partitioning problem in which the min-cut criteria is to be satisfied for page boundaries. The edges of the graph are weighted using tension³. The latest studies on static clustering [13, 7], reveal graph partitioning algorithms as the family of clustering algorithms, providing the best quality of clustering. It is for this reason that we have chosen to use greedy graph partitioning (GGP) [7] as the algorithm of comparison.

A large range of buffer replacement algorithms were used in the this study and they include: Least Recently Used (LRU); First In First Out (FIFO); Least Frequently Used (LFU); LRU-K (extends basic LRU by using historical information) [11]; GCLOCK (extension of CLOCK by using training data to help make eviction choices) [10]; belady’s optimal buffer replacement algorithm [2].

3 Problem Statement

Given an object base, a set of workloads that operate on the object base and a buffer replacement algorithm, we are interested in improving the throughput⁴ of the system by reducing the total I/O generated. Periodic rearrangement of objects is permitted as a means of reducing I/O. Database activity may be suspended when rearrangement is taking place. The following assumption can be made: the pattern of object access between successive periods of database operation⁵ bares some degree of similarity.

4 The Case For Cache Conscious Clustering

Traditionally clustering has been formulated as a locality enhancement problem [14]. The metric used for locality is the working set size [6]. The working set size for M consecutive page requests is computed as follows: take M consecutive page requests, eliminate duplicates and then compute the cardinality of the resulting set. Smaller resulting cardinality means higher locality. The best existing clustering algorithm resulted from finding a heuristic⁶ solution to minimising

³ Tension uses simple markov chains to compute the probability that two objects can be accessed one after the other.

⁴ Long term average performance.

⁵ Before and after every rearrangement.

⁶ Since the problem was found to be NP complete.

working set size of $M = 2$. This means given two consecutive page requests the aim of clustering is to confine both requests to the same page. This formulation allows clustering to be optimised for cache sizes of *one* page. However, typically caches can hold more than one page.

The fundamental difference between our approach and that of Tsangaris and Naughton's [13] is that we cluster for a cache of N pages, where N is greater than or equal to one. There are two consequences of clustering for a larger cache. The first is a request for an object on a *different page in memory* is now considered to cost the same as a request to an object of the *same page*⁷. Clustering changes from a problem of, how do I prevent the object graph traversal from leaving this page? To one of, how do I prevent the object graph traversal from leaving memory? The second consequence of clustering for a larger cache is that clustering must be buffer replacement aware.

Clustering for cache sizes of greater than one page should meet the following two objectives:

- confine traversals to in memory pages.
- cluster objects that are going to be used furthest away in the future together into the same page.

The first objective is a natural consequence of the fact a request to an object on a *different page in memory* is considered equally as good as a request to an object on the *same page*. The second objective is due to the aim of buffer replacement algorithms: to evict the page that is going to be referenced furthest away in the future. Ideally at every eviction the furthest away in memory objects should be clustered together.

5 Cache Conscious Clustering (C3)

This section details the new cache conscious family of clustering algorithms (C3). C3 was arrived at due to the following observation: the first objective outlined in section 4, namely that of confining traversals to in memory pages can be further divided into two sub-objectives. The two sub-objectives are:

- maximise the number of hot⁸ objects in memory.
- minimise the number of pages that a traversal touches.

The first sub-objective increases the number of buffer hits and thus increases the chances of confining a traversal to in memory pages. The second sub-objective increases the probability that the pages involved in a traversal are memory resident⁹. It is important to note that the two sub-objectives can potentially conflict.

⁷ For simplicity, we neglect the effects of CPU caches.

⁸ Where 'heat' is simply a measure of access frequency.

⁹ Provided that the working set of the application does not fit in memory. Thus only one portion of pages used by the application can fit in memory at any one time.

For example a hot object may reference many cold objects and thus clustering the hot objects with its cold objects better satisfies the second sub-objective. However the first sub-objective is best met when hot pages are purer in terms of hot objects contained ¹⁰.

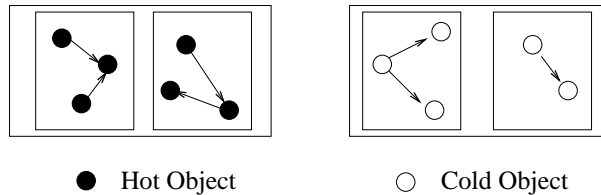
C3's way of meeting the two sub-objectives is to break clustering into two phases. The first phase attempts to satisfy the first sub-objective by dividing objects into regions of similar probability of reference (heat). The second phase attempts to meet the second sub-objective by further dividing objects of each region into pages based on relationships between objects.

Figure 1 gives a diagrammatic representation of one example of the application of our two phased approach. The example shows the first phase dividing the object base into two regions of contrasting heat. The second phase is shown to cluster related objects further into pages.

Phase One



Phase Two



● Hot Object

○ Cold Object

Fig. 1. Diagrammatic description of the two phases of C3. The first phase of this example only divides the objects into two regions of contrasting heat. However in general the first phase of C3 allows the object base to be divided into N regions.

The first phase is accomplished by sorting the objects into decreasing heat. The sorted sequence is then cut at $N - 1$ places to form N regions of more homogeneous heat. The places at which the $N - 1$ cuts occur is a property of the particular C3 algorithm. See section 6 for an example. The way the sorted sequence of objects is cut into regions has a large effect on the performance of the clustering algorithm. Cutting the sorted sequence into smaller regions has

¹⁰ Since most buffer replacement algorithms try to keep hot pages in memory.

the following consequences: regions with objects of more homogeneous heat are created; the probability that two related objects are placed into the same region is decreased. The first consequence is beneficial to the first sub-objective, since it creates purer hot pages. However the second consequence runs counter to the second sub-objective, since related objects of one traversal are more likely to be assigned to different pages. Therefore, the sorted sequence must be cut carefully.

The second phase further clusters each heat region into pages using any clustering algorithm that clusters based on some notion of relatedness between objects. Clustering algorithms that can be used in the second phase include algorithms that use the following statistics to model user behavior: structural relationships between objects [12, 3]; weighted structural relationships [12]; tension [13, 7].

Both phases of C3 contribute to C3 satisfying the second objective of section 4, namely clustering objects that are going to be used furthest away in the future together into the same page. The first phase of C3 removes potential hot objects from cold pages¹¹. This is beneficial since hot objects are potentially more likely to be used in the near future. Thus their removal from cold pages increases the probability that cold pages are referenced further away in the future. The second phase further groups cold objects that are likely to be used in the same time interval, together. Thus the combination of the two phases results in cold pages with many cold objects that are likely to be used in the same time interval, being placed together. This allows buffer replacement algorithms to select pages that contain a larger fraction of the furthest referenced objects, when performing eviction.

6 GPC3 a Concrete Example of C3

This section outlines the GPC3 clustering algorithm. GPC3 is a concrete example of a C3 clustering algorithm. In the first phase GPC3 divides the object base into two regions of contrasting heat. The cut occurs at the point that results in all objects in the hot region just fitting in memory. The second phase of C3 uses the graph partitioning algorithm GGP [7] to further divide the objects in each region into pages.

The second phase of GPC3 is particularly effective for reducing I/O of the cold region. Graph partitioning the cold region increases the locality of reference of cold pages. Since cold pages are not expected to stay in memory long it is appropriate to formulate the clustering of cold objects as a WSS with $M = 2$, minimisation problem. This effectively means objects belonging to the cold region are clustered for a cache of size one. Graph partitioning algorithms are the best algorithms for solving the WSS with $M = 2$, minimisation problem [13, 7].

¹¹ Cold pages are more likely to be targeted for eviction, since “the purpose of buffer replacement algorithms is to keep popular pages memory resident” [11].

7 Experimental Results

In this section we present results of experiments conducted with the Object Clustering Benchmark (OCB) [4] using the Virtual Object Oriented Database simulator, (VOODB) [5]. VOODB is based on a generic discrete-event simulation framework. Its purpose is to allow performance evaluations of OODBMs in general, and optimisation methods like clustering in particular.

OCB is designed to benchmark OODB systems and clustering policies in particular. The OCB database has a variety of parameters which make it very user-tunable. A database is generated by setting parameters such as total number of objects, maximum number of references per class, base instance size, number of classes, etc... Once these parameters are set, a database conforming to these parameters is randomly generated. The database consists of objects of varying sizes. In the experiments conducted in this paper the objects varied in size from 50 to 1200 bytes and the average object size was 268 bytes. A total of 20,000 objects were used, resulting in a database size of 5.3 MB. The default OCB parameters were used, with the only exception being the number of class was set to 50. The changes to the default parameters of VOODB, include: system class set to centralised; multiprogramming level set to 1; object initial placement set to sequential.

The results were generated via three steps. The first *training* step runs the database and collects statistical data of object access. The second *clustering* step uses the training data with the clustering algorithm to rearrange objects. The third *evaluation* step measures I/O generated from running the workload on the newly clustered database.

Our experiments compare the performance of two clustering algorithms, Greedy Graph Partitioning (GGP) and GPC3. In the experiments the point at which our implementation of GPC3 cuts the object graph was set at three different places. These places were 60%, 90% and 120% of buffer size. This was done to test the sensitivity of GPC3 to variation of hot region size. 90% was chosen instead of 100% since it was found to give better results. This gives a indication of the sensitivity of GPC3 to its hot region size setting. As further work we plan to develop and test ways of generating the optimal hot region size setting, using information gathered in the training step.

The OCB workload used in this study included simple traversals, hierarchical traversals and stochastic traversals [4]. The depth of the traversals were 2, 4, and 6 respectively. 10000 transactions¹² were run for every result reported in this paper.

We introduced skew into the way in which roots of traversals were selected. Roots were broken up into hot and cold regions. In all experiments the hot region was set to 2 % size of database and had a 98 % probability of access¹³. However

¹² Each transaction involved execution of one of the three traversals.

¹³ That is there was a 98 % probability that the root of a traversal is from the hot region.

the number of hot objects generated by this pattern of access is greater than 2 % of database size since each traversal accesses more objects than just the root.

7.1 Dependency Between Clustering and Buffer Replacement

In this section we explore the performance of GGP and GPC3 on ten different buffer replacement algorithms. The ten different buffer replacement algorithms investigated include: random (RAND); First In First Out (FIFO-N); CLOCK (CL-N); traditional Least Recently Used (LRU1-N); GCLOCK (GCL-N); Least Frequently Used (LFU-N); Least Recently Used K algorithm with K set to 2 (LRU2-H); GCLOCK algorithm using training data (GCL-T); Least Frequently Used algorithm using training data (LFU-T); Belady's optimum algorithm (OPT-T). Algorithms with a T suffix uses information gathered in the training step of the experiment to help make more accurate replacement decisions during the evaluation step. The N suffix is used for algorithms that do not use training data and also resets statistics for a page when it is first loaded into memory. Algorithms with a H suffix retains history information for a page when it is evicted from memory. However H suffix algorithms do not use training data.

The results from running the above described experiment on two different buffer sizes are depicted on figure 2. The first general observation is that GPC3 outperforms GGP for all buffer replacement algorithms when the buffer size is 2400KB. However for buffer size of 400KB, GPC3 and GGP performed approximately the same, for nine of the ten buffer replacement algorithms tested. We now explain the degradation in relative performance of GPC3 when buffer size is decreased. At smaller buffer sizes the various buffer replacement algorithms find it more difficult to retain in memory pages, belonging to the hot region of GPC3. This failing means clustering hot objects together is less profitable since even when many hot objects are clustered into the same page, the page still has a high probability of being evicted due to the small buffer size.

The only result in which GPC3 performs substantially worse than GGP is when buffer size is 400 KB and the buffer replacement algorithm used is LFU-N (depicted on figure 2 (b)). The reason for GPC3's bad performance in this situation lies in the way LFU-N works. LFU-N does not favor retaining recently loaded pages in memory. All other algorithms with the exception of RAND favor retaining recently loaded pages in memory.

7.2 Variation of Buffer Size Experiment

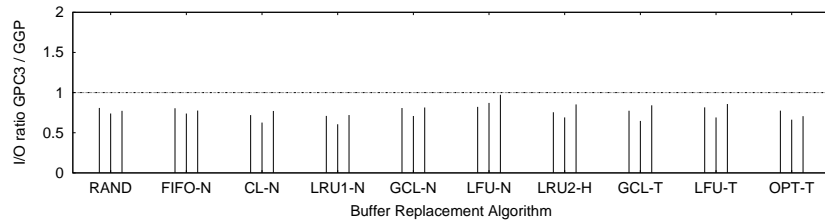
This experiment examined in more detail the effect of buffer size on the performance of GGP and GPC3. The buffer replacement algorithm used for this experiment is the popular LRU1-N algorithm.

The results of this experiment are shown on figure 3, the abbreviation GPC3-x is used to denote the GPC3 algorithm with a hot region size of x fraction of buffer size. These results show GPC3 outperforming GGP for a large region of buffer sizes (500 KB to 4200 KB). The reason for GPC3 to perform worse than or equal to GGP when the buffer size is smaller than 500 KB is again the

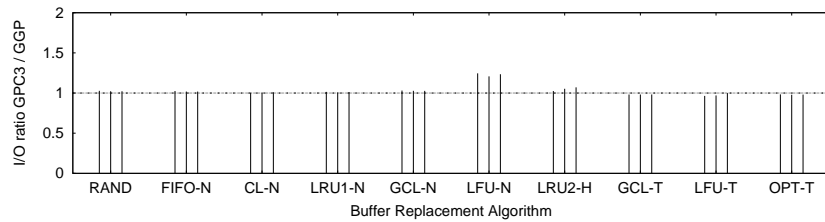
fact at smaller buffer sizes the buffer replacement algorithm can not keep many hot pages in memory. The inability to keep hot pages in memory degrades the performance of GPC3, as explained in section 7.1.

When comparing GPC3 with various settings, on figure 3 (b), a hot region size of 0.9 fraction of buffer size performed best, once the buffer size was larger than 1800 KB. This result can be explained by the fact, once the buffer is large enough (thus giving the buffer replacement algorithm a good chance to keep hot pages memory resident), then it is advantageous to be able to fill almost all of the cache with hot pages (as is the case when hot region size is set to 0.9). When hot region is set to 1.2, the hot region is too large to fit entirely into memory and thus performance degrades.

When the buffer size is greater than 4200 KB all clustering algorithms perform the same, since at these buffer sizes the entire working set can fit in memory.



(a) Buffer size of 2400 KB



(b) Buffer size of 400 KB

Fig. 2. A performance comparison between GGP and GPC3 for various buffer replacement algorithms. The results depict the I/O ratio of GPC3 over GGP. A result of less than one indicate GPC3 produces less I/O than GGP. A result of 0.5 indicates GPC3 only producing half as many I/Os as GGP. Three results are reported for each buffer replacement algorithm. These results differ in the hot region size of GPC3, 60%, 90%, and 120% size of buffer respectively. The geometric mean across all the buffer replacement algorithms for buffer size of 400KB, are 1.028, 1.024 and 1.034 respectively. Geometric mean for buffer size of 2400KB, are 0.778, 0.731 and 0.806 respectively.

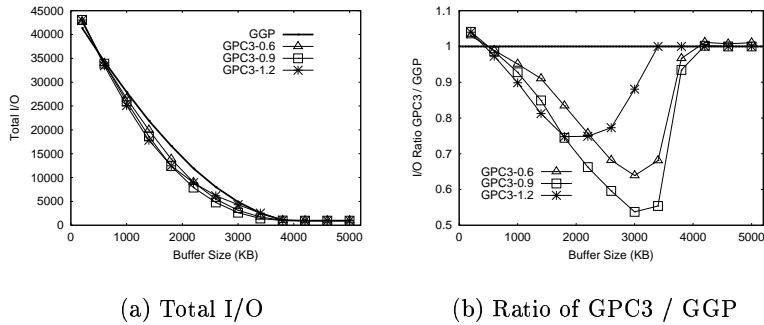


Fig. 3. The effects of variation of buffer size on the performance of clustering algorithms.

8 Conclusion

This paper has highlighted the importance of cache consciousness for clustering algorithms. The clustering problem has been restated for the first time to incorporate caching. As a result a new family of clustering algorithms (C3) have been developed. In order to test the performance of C3, GPC3 a concrete example of C3 was proposed.

A performance evaluation of GPC3 was conducted against GGP. GGP is a member of the graph partitioning family of clustering algorithms, which has been shown to provide the best quality of clustering. Results indicate GPC3 outperforms GGP when the buffer size is large compared to the database size and about the same performance for smaller buffer sizes. This implies systems with a large memory size compared to the working set size will benefit more from GPC3.

9 Further Work

An obvious area for further work is to introduce more algorithms that use the C3 approach of clustering. Intelligent ways of finding points at which to cut the sorted sequence of objects in phase one of C3 is an interesting area of further research.

The cache conscious approach to clustering should be introduced into dynamic clustering¹⁴. A possible way of doing this is to apply OPCF [8] on C3. OPCF is a generic framework by which static clustering algorithms can be transformed into dynamic algorithms.

¹⁴ In dynamic clustering, clustering occurs concurrently with database operation.

Acknowledgement

We would like to thank Jerome Darmont for making the sources of VOODB and OCB freely available. These tools have helped tremendously for our experimental work. In addition we would like to thank John Zigman and Stephen Blackburn for their helpful comments and suggestions.

References

1. AILAMAKI, A., DEWITT, D. J., HILL, M. D., AND WOOD, D. A. DBMSs on a modern processor: Where does time go? In *The 25th VLDB conference, Edinburgh, Scotland* (September 1999), pp. 266–277.
2. BELADY, L. A. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal* 5, 2 (1966).
3. BENZAKEN, V., AND DELOBEL, C. Enhancing performance in a persistent object store: Clustering strategies in *o2*. Tech. Rep. 50-90, Altair, August 1990.
4. DARMONT, J., PETIT, B., AND SCHNEIDER, M. Ocb: A generic benchmark to evaluate the performances of object-oriented database systems. In *International Conference on Extending Database Technology (EDBT)* (Valencia Spain, March 1998), LNCS Vol. 1377 (Springer), pp. 326–340.
5. DARMONT, J., AND SCHNEIDER, M. VOODB: A generic discrete-event random simulation model to evaluate the performances of oodbs. In *The 25th VLDB conference, Edinburgh, Scotland* (September 1999), pp. 254–265.
6. DENNING, P. J. The working set model of program behavior. *Communications of ACM* 11, 5 (May 1968), 323–333.
7. GERLHOF, C., KEMPER, A., KILGER, C., AND MOERKOTTE, G. Partition-based clustering in object bases: From theory to practice. In *Proceedings of the International Conference on Foundations of Data Organisation and Algorithms (FODO)* (1993).
8. HE, Z., MARQUEZ, A., AND BLACKBURN, S. Opportunistic prioritised clustering framework (OPCF). In *Symposium on Objects and Databases* (June 2000).
9. KNAFLA, N. *Prefetching Techniques for Client/Server, Object-Oriented Database Systems*. PhD thesis, University of Edinburgh, 1999.
10. NICOLA, V. F., DAN, A., AND DIAS, D. M. Analysis of the generalized clock buffer replacement scheme for database transaction processing. In *Proc. of the 1992 ACM SIGMETRICS Conference* (1992), pp. 35–46.
11. O’NEIL, E. J., O’NEIL, P. E., AND WEIKUM, G. The lru-k page replacement algorithm for database disk buffering. In *Proc. of the 1993 ACM SIGMOD Conference*. 1993, pp. 297–306.
12. STAMOS, J. Static grouping of small objects to enhance performance of a paged virtual memory. In *ACM Transactions on Computer Systems* (May 1984), vol. 2, pp. 155–180.
13. TSANGARIS, E.-M. M. *Principles of Static Clustering For Object Oriented Databases*. PhD thesis, University of Wisconsin-Madison, 1992.
14. TSANGARIS, M. M., AND NAUGHTON, J. F. A stochastic approach for clustering in object bases. In *Proceedings of the ACM SIGMOD conference on Management of Data* (1991), pp. 12–21.