On Using Cache Conscious Clustering for Improving OODBMS Performance

Zhen He[1] Richard Lai[1] Alonso Marquez[2]

[1]*Department of Computer Science and Computer Engineering, La Trobe University, Victoria 3086, Australia*
{*z.he,r.lai*}*@latrobe.edu.au*

[2]*Department of Computer Science, The Australian National University Canberra, ACT 0200, Australia*
alonso@cs.anu.edu.au

Abstract

The two main techniques of improving I/O performance of Object Oriented Database Management Systems(OODBMS) are *clustering* and *buffer replacement*. *Clustering* is the placement of objects accessed near to each other in time into the same page. *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 least in the future. These two techniques both influence the likelihood of a requested object being memory resident. We believe an effective way of reducing disk I/O is to take advantage of the synergy that exists between *clustering*, and *buffer replacement*. Hence, we design a framework, whereby clustering algorithms incorporating buffer replacement cache behaviour can be conveniently employed for enhancing the I/O performance of OODBMS. We call this new type of clustering algorithm, *Cache Conscious Clustering (C3)*. In this paper, we present the C3 framework, and a C3 algorithm that we have developed, namely C3-GP. We have tested C3-GP against three well known clustering algorithms. The results show that C3-GP out performs them by up to 40% when using popular buffer replacement algorithms such as LRU and CLOCK. C3-GP offers the same performance as the best existing clustering algorithm when the buffer size compared to the database size is very small.

Keywords: Object-oriented databases, clustering, buffer replacement, caching, database optimization.

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 [18]. 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.

We believe the best way of reducing disk I/O is 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 exists between the two techniques. In addition, exploitation of that synergy results in improvements in performance.

Traditionally static clustering algorithms have generally been designed to place objects likely to be co-referenced into the same page [25, 2, 12, 10]. On the surface this seems like a reasonable approach, since it confines object graph traversals as much as possible to one page. By minimising the likelihood of traversing out of the current page, the chances of requiring a disk load are minimised. However, upon closer inspection we can criticise this approach as being too conservative. This is because the assumption that navigations out of the current page have a high probability of causing a page load is only valid when either the cache size is one page or the cache size is larger but the buffer replacement algorithm only keeps pages cache resident for very short durations. This papers aims to demonstrate simple synergistic modifications to existing algorithms can result in improved performance. To this end we exploit our knowledge of buffer replacement algorithm behaviour to design static clustering algorithms that tolerate navigations out of the current page so long as the navigation proceeds into another cache resident page. We term this new approach cache

conscious clustering(C3). In order to make our approach more generally applicable we created the C3 framework. The C3 framework produces a *family* of static clustering algorithms that all possess the property of cache consciousness.

Experimental results show a C3 algorithm called C3-GP outperforms the three existing static clustering algorithms, probability ranking principle algorithm (PRP) [25], greedy graph partitioning (GGP) [12], and Wisconsin greedy graph partitioning WGGP [25] in a variety of situations. We believe an effective way of reducing disk I/O is to take advantage of the synergy that exists between *clustering*, and *buffer replacement*. Hence, we design a framework, whereby clustering algorithms incorporating buffer replacement behaviour can be conveniently employed for enhancing the I/O performance of OODBMS. We call this new type of clustering algorithm, *Cache Conscious Clustering (C3)*.

In this paper, we present the C3 framework, and a C3 algorithm that we have developed, namely C3-GP. We have tested C3-GP against three well known clustering algorithms. The main contribution of this paper is the development of the C3 framework that incorporates buffer replacement behaviour into clustering. Cache conscious clustering algorithms can be developing using this framework. A preliminary version of this paper was presented at the 12th International Conference on Database and Expert Systems Applications (DEXA 01)[14]. This paper extends our preliminary work in the following ways: the framework to include an extra phase which allows users to define their own metric for regularity of object references; the simulation to include comparisons with two additional clustering algorithms (WGGP, PRP); and three more simulation results, which allow us to gain a better understanding of the performance trade-offs of the proposed algorithm.

2 Related Work

Although there is much existing literature on both static clustering [25, 2, 12, 10] and buffer replacement [20, 21, 3, 17, 22, 19], to our knowledge no prior work explores the synergies between the two techniques. In this section we first review existing clustering algorithms. Second, we review existing buffer replacement algorithms. Third, we review work related to object placement in generational garbage collectors.

2.1 Static Clustering Algorithms

Most static clustering algorithms use as input a graph representation of the access patterns (called clustering graph or CG). Static clustering algorithms can be classified via the clustering graph used. There are three typical types of clustering graphs:

- *Object Graph (OG):* In this approach the object graph itself is used as the sole source of information for clustering purposes. Static clustering algorithms that take this approach include the depth first search (DFS) [24], breath first search (BFS) [24] and placement tree (PT) [4] algorithms. The object graph approach does not encapsulate information regarding navigations that do not follow the object graph and it also does not weight more frequently traveled paths of the object graph higher than the less frequently traveled. However, the advantage of this approach is reduced statistic overheads.
- *Statistical Object Graph (SOG):* In this approach, the object graph is weighed. The nodes are weighed according to frequency of object access and edges are weighted based on frequency of edge traversal. Two static clustering algorithms that use SOG are the weighted depth first search (WDFS) [24] and the cactus clustering algorithm [10]. The drawback of using SOG is its inability to model object co-references that do not follow the object graph.
- *Simple Markov Chain Model (SMC):* In this approach, the directed graph form of a first order Markov model is used as the clustering graph. Each accessible object in the system is represented by a node in the graph. Any positive probability that one object can be accessed after some other object, is represented as a directed edge. The node weights are labeled by probabilities of accessing the object and the edge weights are labeled by the transition probabilities. The algorithms that use this approach include: the probability ranking principle algorithm (PRP) [25] (which only uses the node weights of the SMC model); the Wisconsin greedy graph partition algorithm (WGGP) [25]; the Kernighan-Lin graph partitioning algorithm (KL) [25]; and the greedy graph partitioning algorithm [12].

We describe the PRP, GGP, and WGGP clustering algorithms in more detail. These three algorithms are used in our simulation study. The PRP algorithm simply sorts all objects according to decreasing heat (where 'heat' is simply a measure of access frequency). Then the objects are just placed into pages in this presorted order.

GGP approaches the problem of clustering as a graph partitioning problem, where the min-cut criteria is to be satisfied. The clustering graph is setup using SMC model (described above). The algorithm attempts to satisfy the min-cut criteria by first assigning only one object to a partition and then combining partitions in a greedy manner. GGP does not require objects to be relatively uniform in size and also places no restrictions on the configuration of the clustering graph (eg. graph must be acyclic). WGGP, like GGP is also a greedy graph partitioning algorithm, however

it creates partitions in a different way. WGGP first sorts all objects in heat order. The algorithm starts by placing the hottest object into the first partition and then incrementally places the remaining objects as follows. Among all the objects that will fit into the current partition (partition size must be smaller than a page), find the object that have not been placed and have the highest edge weight (i.e. edge weight of the clustering graph, using SMC to model object level transitions). with the current partition. Place the selected object into the current partition. Repeat until no candidate objects can be found. Start a new partition using the hottest yet to be placed object as the first object and repeat the entire process.

2.2 Buffer Replacement Algorithms

There have been numerous buffer replacement algorithms presented in the literature [20, 21, 3, 17, 22, 19]. They mainly differ in the way they collect access information. The Least Recently Used (LRU) based policies measure the length of time between successive references of a page. The elapsed time between successive references of a page give a indication of when the page will next be referenced. A page with larger re-reference intervals is less likely to be needed in the near future, and thus would make a better candidate for eviction.

LRU-K extends the basic LRU by recording historical information. The times of the K previous references are recorded. In addition, LRU-K removes the effects of correlated references. Correlated references occurs when the same page is re-referenced in quick succession. LRU-K's historical reference information and removal of correlated references gives it superior performance compared to the basic LRU.

Frequency based techniques like Least Frequently Used (LFU), make eviction decisions based on frequency of reference information. The assumption made by these algorithms is that pages that are referenced more frequently are more likely to be re-referenced in the near future. Thus the page selected for eviction is the least frequently used page.

There are other simple algorithms that are designed to collect and store a small amount of information for performance reasons. These include, the First In First Out (FIFO) and CLOCK algorithms. The CLOCK algorithm is a popular replacement algorithm because of its simplicity and its ability to approximate the performance of the Least Recently Used (LRU) replacement policy. The CLOCK algorithm have been extended by GCLOCK [20]. GCLOCK uses a circular buffer and a weight associated with each page brought into buffer to decide which page to replace. The weights may be different for different data types. The performance of GCLOCK can be better than LRU and performances very close to optimal can be achieved for some situations.

Finally there is Belady's optimal buffer replacement algorithm [3]. This algorithm makes replacement decisions based on analysis of the entire reference trace. Therefore this algorithm can not be practically applied to real OODBMSs, however it is a useful tool of comparison.

2.3 Object Placement in Generational Garbage Collection

There has been existing work in generational garbage collection in the area of grouping hot objects or object fragments together in CPU cache lines in order to minimize cache line misses[6, 16, 23]. These algorithms differ from our approach in that they are designed for exploiting the inherent moving of heap objects in a generational garbage collector for minimizing CPU cache line misses rather than the off-line re-arrangement of in-memory objects to minimize buffer misses. Since our algorithm works off-line we can use more CPU intensive clustering algorithms to obtain a more optimal clustering.

3 Assumptions and Definitions

3.1 Assumptions

The work in this paper makes the following assumptions:

1. The entire database can be shut off for rearrangement to take place.
2. Patterns of object access between rearrangements bear some degree of similarity.
3. All objects are smaller than one page in size. Since large objects(larger than one page in size) do not benefit from clustering, we choose to focus our study on objects smaller than a page in size. However, the techniques in this paper can still be applied when large objects are present. For example, large objects can be placed in a separate area of the object store and dynamic clustering algorithms can ignore them.
4. Objects are moved and mapped from one consistent state to another.

4 C3 Framework

Our C3 framework allows a family of cache conscious clustering algorithms to be defined. This gives researchers the ability to create new cache conscious clustering algorithm by extending the ideas presented in this paper.

4.1 Framework Objective

Customising static clustering for every possible buffer replacement technique is beyond the scope of this paper. However, buffer replacement algorithms in general aim to retain in memory pages that are likely to be needed in the near future. Thus a static clustering algorithm that improves *locality* of the client reference stream should perform well for most buffer replacement algorithms. WSS[9, 26] is a metric that measures locality. We thus aim to find static clustering algorithms that minimise WSS.

WSS is first used within the static clustering context by [25]. A formal definition of WSS follows. $WSS(w)$ is the expected cardinality of the set $R_t^{(w)}$ of w consecutive page requests starting at time t. That is: take these w page requests, eliminate duplicates, and compute the cardinality of the resulting set. Note that the larger the cardinality, the fewer the duplicates, hence the lower the locality. $WSS(w)$ can be described mathematically as:

$$WSS(w) = E(\| R_t^{(w)} \|) \tag{1}$$

$1 \leq WSS(w) \leq w$. If $w > M$ then the upper limit for $WSS(w)$ is M (where M is the number of pages the whole object space maps to). The window parameter w allows us to optimise for different cache sizes, because the smaller $WSS(w)$ is, the more effective a cache size of w is. If the cache size is $\geq M$, any replacement policy will work since the whole object base fits in memory.

For the IID and time invariant Markovian models (including the SMC model used in this paper), the t subscript can be dropped.

Tsangaris [25] first formulated static clustering as a minimisation of the WSS metric. However, they by-passed the synergy between static clustering and buffer replacement via two decisions. Firstly they only optimise for $WSS(2)$. 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. This is a boundary case and one in which buffer replacement is irrelevant. However, this approach still managed to inspire the graph partitioning family of clustering algorithms which offer best performance among all existing algorithms[25]. Second, [25] did not test the effect of different buffer replacement algorithms. In fact, only one experiment uses a metric that is effected by buffer replacement algorithms.

The fundamental difference between our approach and that of [25] is that we cluster for $WSS(w)$, where $w \geq 1$. Thus we cluster for caches larger than one page, where buffer replacement behaviour is relevant. We term static clustering algorithms that solve the $WSS(w), w \geq 1$ problem *cache conscious* clustering algorithms.

The objective of the C3 framework is to produce a family of clustering algorithms which are all heuristic solutions to the following problem: minimise $WSS(w)$ for $w \geq 1$.

4.2 Framework Definition

The C3 framework minimises $WSS(w)$ by attempting to meet the following sub-objectives:

- group regularly referenced objects into the same pages.
- group non-regularly referenced objects that are to be used at the same time into the same page.

The first sub-objective produces pages with a high concentration of regularly referenced objects. This ensures regularly referenced objects are not spread across a large number of pages. Spreading regularly referenced objects across many pages reduces the number of duplicate references occurring in any sequence of w page requests. In contrast, regularly referenced pages produced from a high concentration of regularly referenced objects are likely to produce many duplicate references for any sequence of w page requests. A large number of duplicate references minimises the $WSS(w)$ metric.

The second sub-objective produces pages that are heavily referenced for short periods of time and then not referenced for long periods of time. This is because we assume non-regularly referenced objects have long non-reference periods. If groups of non-regularly referenced objects that have similar reference times are placed in the same page then the entire page will have long non-reference periods and short periods of heavy reference. This sub-objective thus produces pages that get many successive duplicate references. Thus this approach minimises the $WSS(w)$ metric.

C3 uses the following three steps to meet the sub-objectives described above:

1. *Define metric for regularly referenced objects.* This step provides a means of ranking objects based on how regularly they will be referenced. The best definition of this metric is likely to depend on the characteristics of the trace. For example a trace that contains approximately uniform object access intervals can use a simple metric like heat (the number of times the object has been referenced) as its regularity metric. A trace with non-uniform access behaviour may use the average time between references (smaller average time indicates higher regularity).
2. *Separate into N regions of homogeneous regularity.* This step divides objects into regions of similar regularity of reference. This is accomplished by sorting the objects into decreasing regularity of reference. The sorted sequence is then cut at N - 1 places to form N regions of more homogeneous regularity. The places at which the N - 1 cuts occur is a property of the particular C3 algorithm. See section 5 for an example.
3. *Cluster each region based on relatedness separately.* This step further divides each regularity region into pages using any clustering algorithm that clusters based on some notion of relatedness between objects. Most static clustering algorithms mentioned in section 2.1 fall into this category.

The combination of the three steps produces a group of pages that have a high concentration of regularly referenced objects and thus complies with the first sub-objective. The second group of pages produced are non-regularly referenced objects that have been clustered based on relatedness and thus complies with the second sub-objective.

In the second step, 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 the following consequences: regions with objects of more homogeneous regularity are created; and the probability that two related objects (objects that are referenced one after the other) are placed into the same region is decreased. The first consequence is beneficial for creating pages that comply with the first sub-objective, since it creates pages with a higher concentration of regular objects. However, the second consequence is detrimental for creating pages that comply with the second sub-objective. This is because related objects (which are likely to be used at the same time) are more likely to be assigned to *different* pages. Therefore, the sorted sequence must be cut carefully.

5 Cache Conscious Graph Partitioning

In this section we describe C3-GP, an concrete algorithm produced using the C3 framework. C3-GP makes the following framework decisions:

1. *Define metric for regularly referenced objects.* C3-GP uses heat as the regularity metric. It assumes a uniform random reference distribution. That is, it assumes objects referenced with high regularity will have a higher total number of references than objects which are not accessed regularly. It is clear this assumption does not always hold in real life, since sometimes an object may not be referenced regularly but when it does get referenced it is hit many times. The search for the best clustering algorithm for every case is beyond the scope of this paper. Extending the work in this paper to more general cases is an interesting area of future work.
2. *Separate into N regions of homogeneous regularity.* C3-GP divides the object base into two regions of contrasting regularity. It labels the region with high regularity the 'hot' region, and the other region the 'cold' region. The cut occurs at the point that results in all objects in the hot region just fitting in memory.
3. *Cluster each region based on relatedness separately.* C3-GP uses the graph partitioning algorithm GGP GGP to further divide the objects in each region into pages. This step 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 $WSS(2)$ 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 $WSS(2)$ minimisation problem [25, 12].

The time complexity of C3-GP is $O(N\ log\ N\ +\ E_h\ log\ E_h\ +\ E_c\ log\ E_c)$, where N is the number of objects in the entire object space. E_h and E_c are the number of edges in the clustering graph of the hot and cold regions respectively. The time complexity is determined by the sorting of all objects in the entire object space according to heat, sorting of clustering graph edge weights of the hot region and sorting of clustering graph edge weights of the cold region, respectively.

6 Simulation Setup

The simulations are conducted using the Object Clustering Benchmark (OCB) [7] and the Virtual Object Oriented Database simulator, (VOODB) [8]. 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 OODBM systems and clustering polices 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 simulations 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 are used, resulting in a database size of 5.3 MB.

The parameters of OCB and VOODB used to conduct the simulations in this paper are specified in table 1. VOODB parameters involving time have been omitted from this table, since the results reported are in terms of I/O performance.

(a) OCB database parameters

Parameter Description	Value
number of classes in the database	50
maximum number of references, per class	10
instances base size, per class	50
total number of objects	20000
number of reference types	4
reference types random distribution	uniform
class reference random distribution	uniform
objects in classes random distribution	uniform
objects references random distribution	uniform

(b) VOODB parameters

Parameter Description	Value
system class	centralised
disk page size	4096 bytes
buffer size	varies
buffer replacement policy	varies
pre-fetch policy	none
multiprogramming level	1
number of users	1
object initial placement	sequential

Table 1. Parameters used for OCB and VOODB.

The results are 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.

In this paper we compare the performance of the C3 algorithm C3-GP with three existing static clustering algorithms. The three existing static clustering algorithms are the probability ranking principle algorithm (PRP) [25], greedy graph partitioning (GGP) [12], and Wisconsin greedy graph partitioning WGGP [25]. The reason for choosing these algorithms is they all use the SMC clustering graph. The SMC clustering graph algorithms has been shown by [25] to give best general performance.

The static clustering algorithms shown on the graphs are labeled as follows:

- **NC**: No Clustering;
- **PRP**: Probability Ranking Principle;
- **WGGP**: Wisconsin Greedy Graph Partitioning;
- **GGP**: Greedy Graph Partitioning;
- **C3-GP**: C3 greedy graph partitioning.

In the simulations we set C3-GP's hot region size parameter to 90% of main memory (with the exception of one simulation in which we investigate the effect of varying hot region size of C3-GP). This is because we found C3-GP performs best when we set its hot region size parameter to 90%.

The results are 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.

The OCB workload used in this study included simple traversals, hierarchical traversals and stochastic traversals [7]. The depth of the traversals are 2, 4, and 6 respectively. 10000 transactions are run for every result reported in this paper. Each transaction involved execution of one of the three traversals.

We introduced skew into the way in which roots of traversals are selected. Roots are broken up into hot and cold regions. In all simulations the hot region was set to 2 % size of database and had a 98 % probability of access (i.e.

there was a 98 % probability that the root of a traversal is from the hot region). However 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. These settings are chosen to represent typical database application behaviour. Gray at. el.[13] cites statistics from a real videotext application in which 3% of the records got 80% of the references. Carey at. el.[5] use a hot region size of 4% in the HOTCOLD workload have been used to measure data caching trade-offs in client/server OODBMSs. Franklin at. el.[11] use a hot region size of 2% in the HOTCOLD workload used to measure the effects of local disk caching for client/server OODBMSs.

The evaluation metric used is total I/O. In the simulations total I/O equals the total transaction read I/O. The reason for using total I/O instead of WSS as our evaluation metric is that ultimately we are interested in how well the algorithms can reduce total I/O. In this paper the WSS metric is only used as a guide to explain the intuitions that led to the design of our algorithms.

7 Simulation Results

In this section we report the results of simulations comparing the performance of three existing highly competitive static clustering algorithms (PRP, WGGP, GGP) with the C3 produced C3-GP algorithm.

7.1 Varying Buffer Size

In this section we report the effects of varying buffer size on the performance of the static clustering algorithms. The buffer replacement algorithm used is the LRU algorithm. The results are shown on Figure 1. The general observation is that C3-GP always performs better than or as well as the existing algorithms. C3-GP outperforms all existing algorithms between buffer sizes of 0.5MB and 5.8MB and shows equal performance for other buffer size settings. The reason for C3-GP performing the same as GP when the buffer size is less than 0.5MB is that at this smaller buffer size the buffer replacement algorithms find it difficult to retain pages belonging to the hot region of C3-GP in memory. This failure 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. When the buffer size is larger than 5.8MB almost all of the active portion of the database fits in memory and thus all static clustering algorithms perform about the same.

At its best C3-GP produces 42% less I/O than GGP (when buffer size is 2.4MB). The performance advantage can be attributed to C3-GP's ability to retain hot objects in memory by creating hot pages with high concentrations of hot objects. This avoids thrashing of pages containing hot objects.

PRP's poor performance at buffer sizes below 6.6MB can be attributed to the fact that it does not attempt to cluster based on object transition information. However, when the buffer size is large enough to fit in the entire active portion of the datatbase (beyond 6.6MB), it performs just as well as the other algorithms. This is because it is just as effective as the other algorithms at mapping the entire active portion of the database into a minimum number of pages.

7.2 Varying Buffer Replacement Algorithm

In this section we explore the performance of the clustering policies: no clustering, PRP, WGGP, GGP and C3-GP 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) [20]; Least Frequently Used (LFU-N); Least Recently Used K algorithm with K set to 2 (LRU2-H) [21]; GCLOCK algorithm using training data (GCL-T); Least Frequently Used algorithm using training data (LFU-T); Belady's optimum algorithm (OPT-T) [3]. Algorithms with a T suffix use information gathered in the training step of the simulation 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 reset 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 of using 1MB and 4MB buffer sizes are reported on Figure 2 (a) and (b) respectively. The results for five different static clustering policies are reported for each buffer replacement algorithm result. The static clustering results are reported in the following order, no clustering, PRP, WGGP, GGP and C3-GP. The results show that for the 1MB buffer size case, C3-GP offers best performance for all buffer replacement algorithms used. When the buffer size is 4MB, C3-GP is the best performer for 8 of the 10 buffer replacement algorithms used. The only cases in which C3-GP is not the best performer is when the buffer size is 4MB and the LFU-N and LFU-T buffer replacement algorithms are used. This is because at 4MB buffer size, almost all of the pages containing hot objects fit in memory, even when

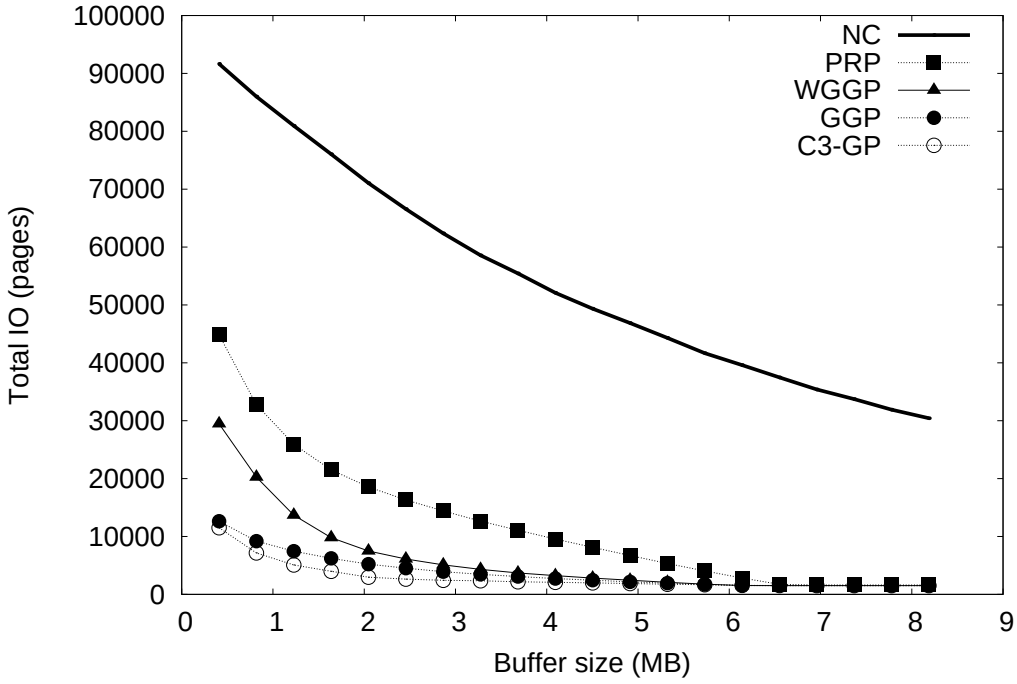


Fig. 1. Comparing the effect of varying buffer size on the total number of IO (pages) for five different static clustering policies. Uses the LRU replacement policy.

the hot objects are spread across many pages (the case with the NC, PRP, WGGP and GGP clustering algorithms). LFU algorithms which keep frequently accessed pages in memory prevents the pages containing hot objects from thrashing. Thus at these settings, C3-GP's ability to prevent thrashing of pages containing hot objects no longer gives it an advantage over the other algorithms. The result is that GGP and WGGP, which cluster solely based on relatedness, are able to meet the second sub-objective of section 4.2 better than C3-GP but do not suffer the negative consequences of not meeting the first sub-objective.

7.3 Varying Database Hot Region Size

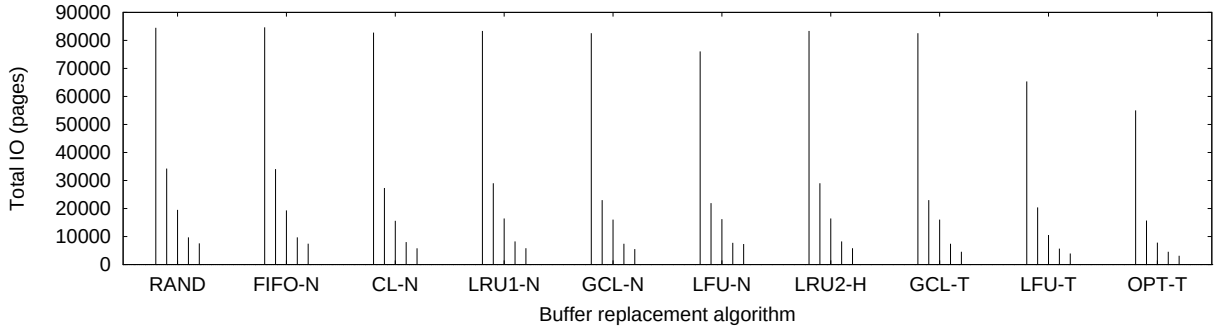
In this simulation we varied the hot region size of the database and kept the probability of hot region access at a constant 0.8. The buffer replacement algorithm used is the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown on Figure 3 (a) and (b). It is encouraging to observe C3-GP offers best performance for both 1MB and 4MB buffer sizes.

When using a buffer size of 1MB, C3-GP's performance lead over GGP diminishes as the database hot region size increases. This is because as the hot region size increases, it becomes increasingly difficult for C3-GP to fit hot objects into *its* hot region. Thus many hot objects end up in cold pages. The result is that C3-GP is no longer able to prevent the thrashing of many of the pages that contain hot objects.

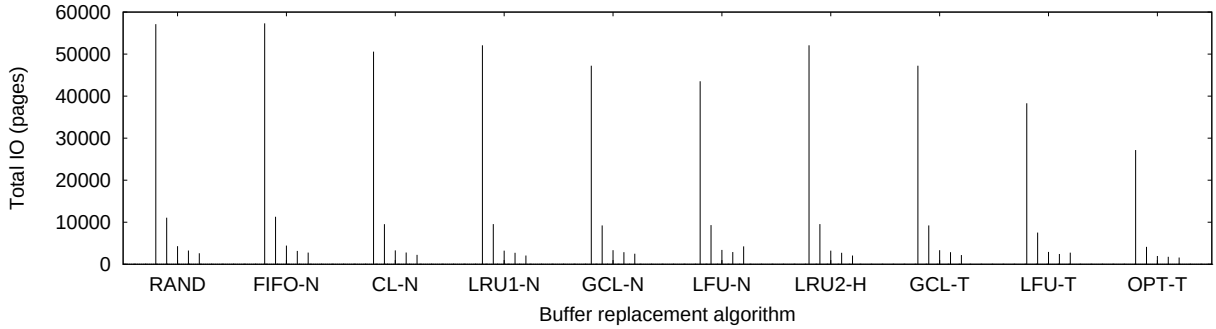
At the large buffer size of 4MB, C3-GP's lead over the other static clustering algorithms increases as the database hot region size increases. The reason behind C3-GP performing about the same as WGGP and GGP at small hot region sizes is that most of the active portion of the database fits in memory at this setting thus most pages containing hot objects are kept in memory even if hot objects are dispersed among many pages (as is the case for WGGP and GGP). However, as the hot region size increases, C3-GP's ability to compact hot objects into fewer pages becomes an increasingly larger advantage when compared to WGGP and GGP.

7.4 Varying Database Hot Region Access Probability

In this simulation we vary the probability of accessing objects in the hot region of the database. The size of the hot region is kept constant at 3% the size of the database. The buffer replacement algorithm used is again the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown in Figure 4 (a) and (b). The results show that C3-GP offers the best performance in general.



(a) 1MB buffer size.



(b) 4MB buffer size.

Fig. 2. Comparing the effect of using different buffer replacement algorithm on total IO (pages) for five different static clustering policies.The results are reported in the following order: no clustering, PRP, WGGP, GGP and C3-GP.

At the 1MB buffer size, C3-GP exhibits the best performance for all the results reported. However, at the 4MB buffer size, C3-GP starts off well in front of the other algorithms but its lead diminishes as the database hot region probability increases. Eventually, at 0.9 all clustering algorithms perform the same. This is because at above 0.9 hot region access probability, almost all queries are confined to the hot region. Since the hot region is relatively small compared to the 4MB buffer size, the entire active portion of the database fits in memory. All of the static clustering algorithms are able to group the active portion of the database together and away from the non-active portion. This explains why all of the algorithms perform the same when the database hot region access probability is above 0.9.

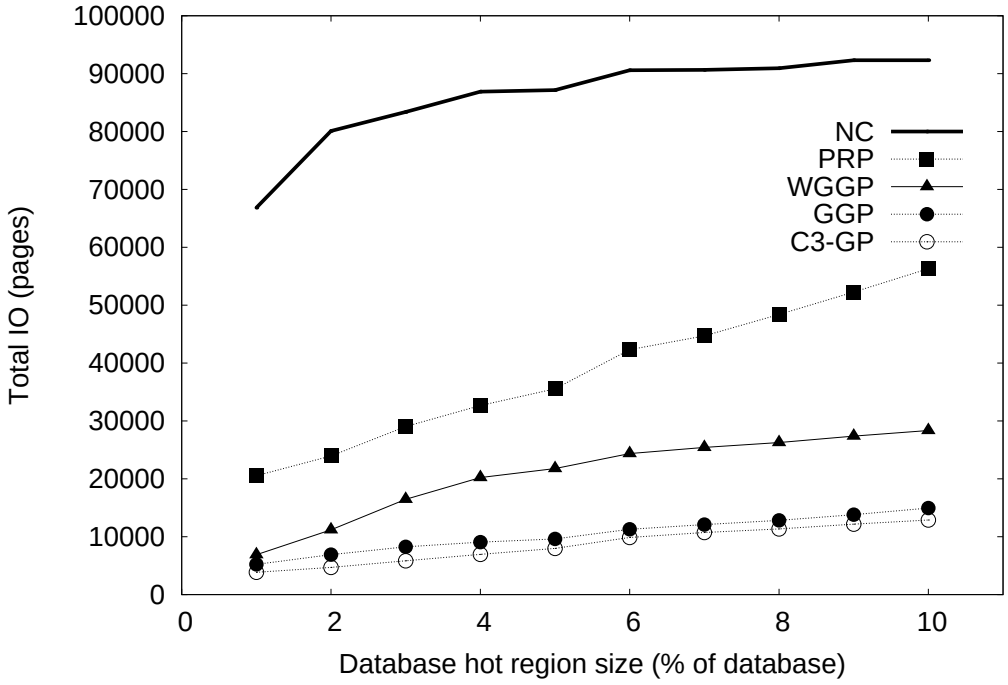
7.5 Varying C3-GP Hot Region Size

In this simulation we varied the size of C3-GP's hot region. Recall from section 5, C3-GP's hot region is created by sorting the objects in decreasing heat and then taking the top x objects as belonging to the hot region. In the definition of C3-GP, x is chosen so that all of the objects just fit into memory. In this simulation we vary the place where the sorted list of objects is cut. The buffer replacement algorithm used is the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown in Figure 5 (a) and (b). The results for NC, PRP, WGGP and GGP do not change when C3-GP hot region size is varied, since these algorithms do not use this parameter.

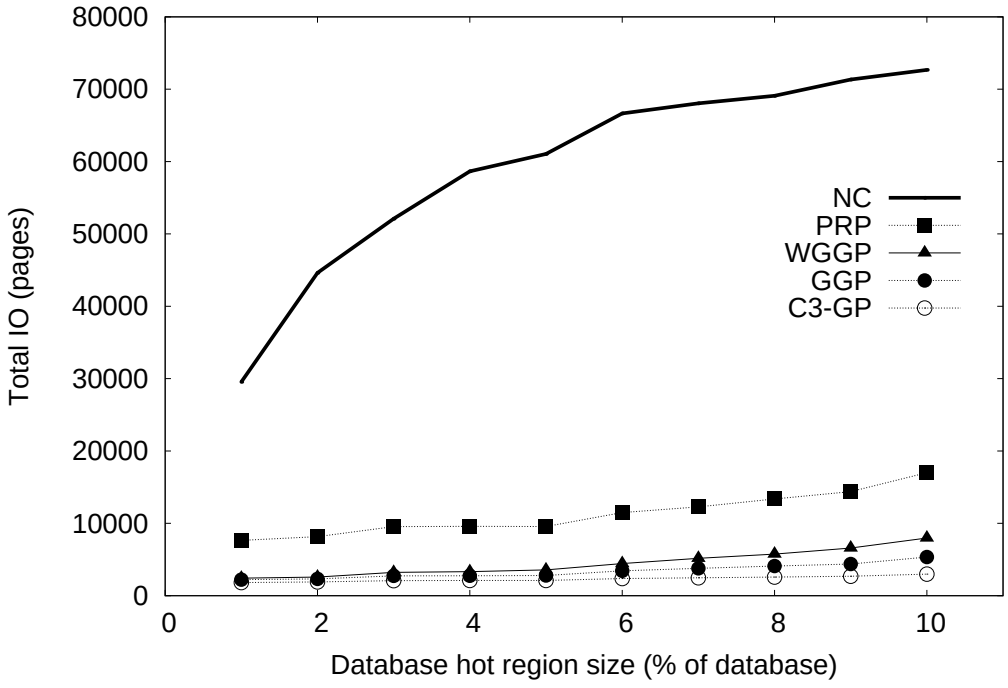
The results show that the optimal C3-GP setting is dependent on the buffer size used. At 1MB buffer size, the optimal setting is approximately 0.9 and at 4MB buffer size the optimal setting is approximately 0.6. This is because the hot region size of the database is the same for both graphs, however the place at which C3-GP divides its hot and cold region is a function of the buffer size, which is different for the two graphs. A possible direction of future work is to develop a method of dividing C3-GP's hot and cold regions based on both the detected database hot region size and the buffer size.

7.6 Training Skew

Until now all of the simulations involved running the same set of transactions for both the training and evaluation steps. In contrast, this simulation explores the effect of running a different set of transactions for the training and evaluation

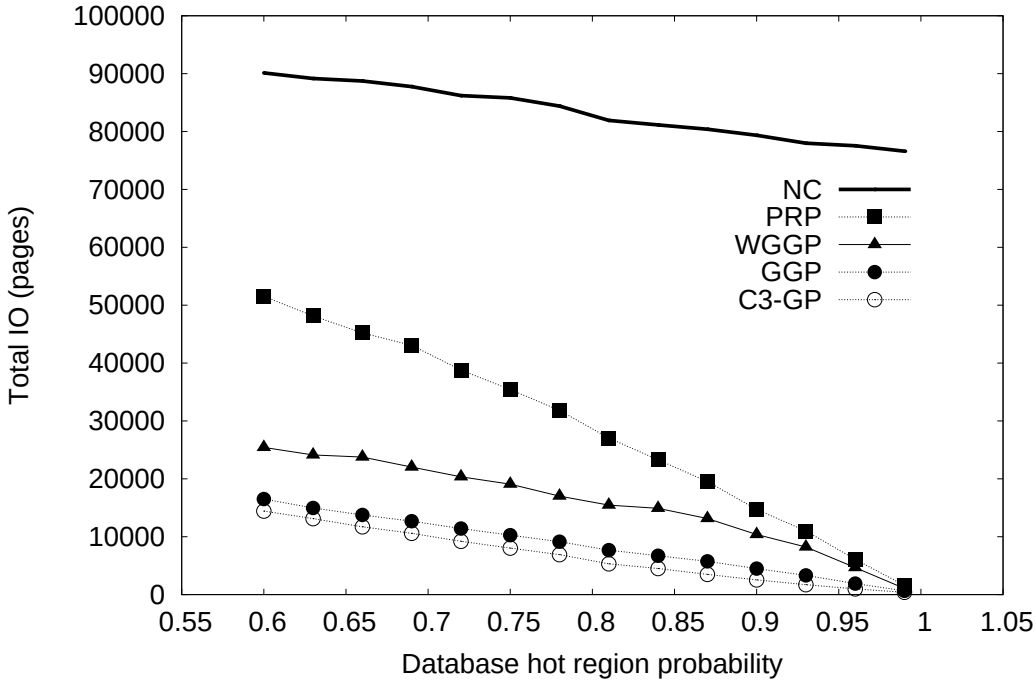


(a) 1MB buffer size.

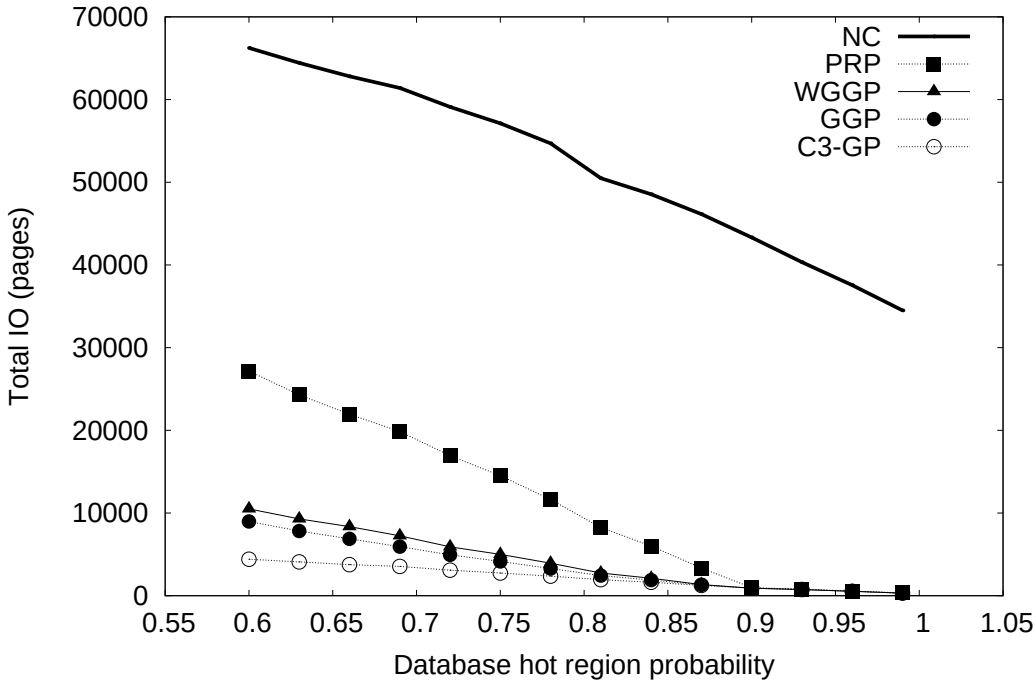


(b) 4MB buffer size.

Fig. 3. Comparing the effect of varying database hot region size on the total IO (pages) for the five different static clustering polices. Uses the LRU replacement policy.

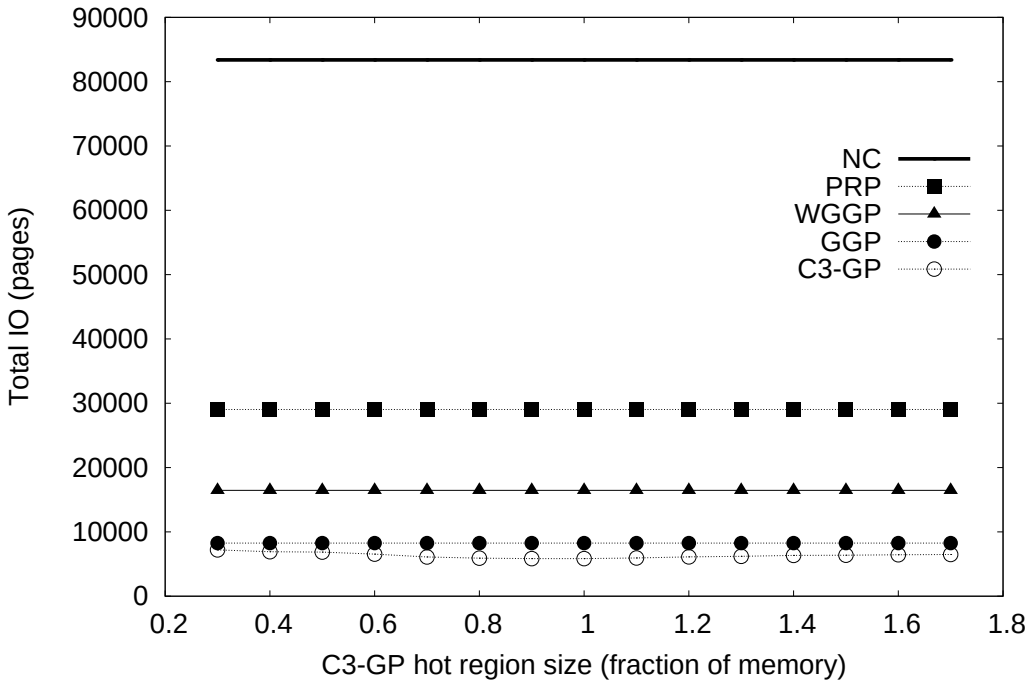


(a) 1MB buffer size.

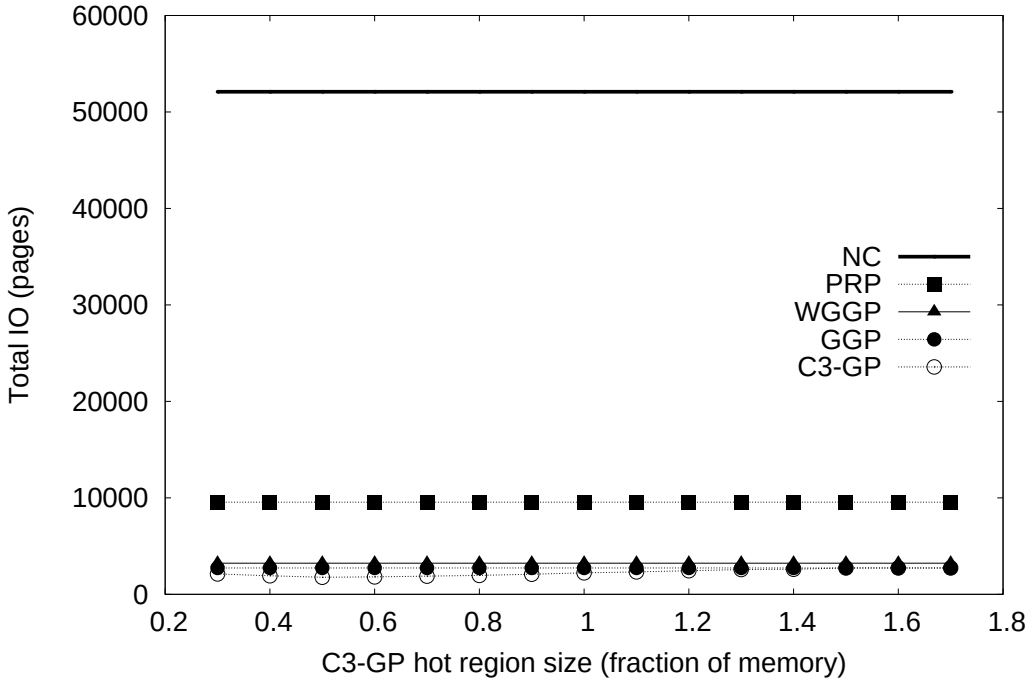


(b) 4MB buffer size.

Fig. 4. Comparing the effect of varying database hot region access probability on the total IO (pages) for five different static clustering polices. Uses the LRU replacement policy.



(a) 1MB buffer size.



(b) 4MB buffer size.

Fig. 5. Comparing the effect on the total IO (pages) when C3-GP's hot region size is varied. Uses the LRU replacement policy.

steps. This is achieved by moving the hot region of the database. The numbers on the x-axis of Figures 6 (a) and (b) show by how much the database hot region is moved. For example, a value of 20% means 20% of the hot region used for the training step became part of the cold region used for the evaluation step. This gives an indication of the degree of difference between transactions used in the training and evaluation steps. The hot region size of database was set to 3% of database size. The buffer replacement algorithm used is the LRU algorithm.

The results show that C3-GP's performance advantage over GGP and WGGP rapidly diminishes as the level of training skew increases. This implies that C3-GP is more sensitive to the quality of training data used. When poor heat information is supplied, C3-GP places hot objects into the cold region and vise versa. The consequence of this behaviour is that C3-GP begins to loose its ability to keep a higher concentration of hot objects in memory. This explains the diminishing of C3-GP's lead over GGP and WGGP when training skew is increased. However, it is encouraging to note that C3-GP's performance never degrades to be worse than GGP or WGGP.

8 Conclusions

In this paper, we have described the C3 framework, whereby clustering algorithms incorporating buffer replacement cache behaviour can be conveniently employed for enhancing the I/O performance of OODBMS. With this framework, a family of clustering algorithms (C3) can be developed. To demonstrate the soundness of the C3 framework, we have developed the C3-GP algorithm.

Our simulation results show that C3-GP out-performs existing static clustering algorithms in a variety of situations. Among the situations tested are 10 different buffer replacement algorithms, various buffer sizes, database hot region sizes, access probabilities, C3-GP's hot region sizes and various amounts of training skew. Among all of the simulation results, C3-GP performed at least as good as the existing algorithms for all but one particular situation (when the LFU-N and LFU-T buffer replacement algorithms are used and the buffer size is large). In particular, G3-GP outperforms GGP (the best existing static clustering algorithm) when the buffer size is large as compared to the database size but offers similar performance when the buffer size is very small. This ability to perform consistently either better or the same as existing algorithms makes C3-GP ideal for deployment in general purpose OODBMS in which workload conditions and system settings are not known a-priori.

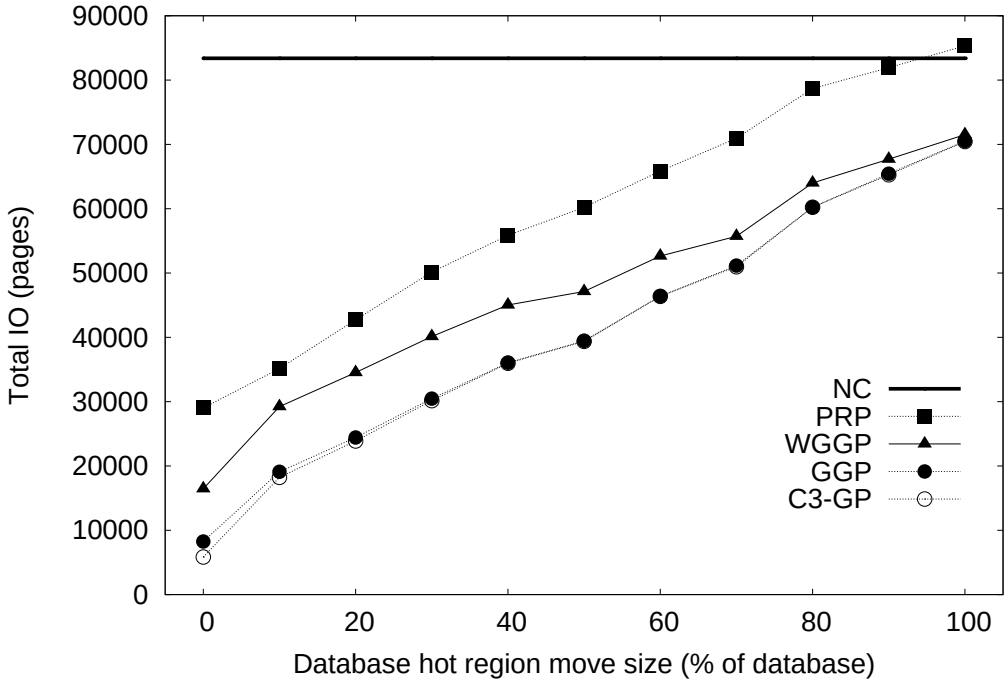
Our simulation study only involved one synthetic benchmark. To further test the validity of our results we will extend our simulation study to involve other object database benchmarks such as OO7 in our future work. We have focused our work on developing *static* clustering algorithm, which are not suitable for use in databases that have frequent access pattern changes or databases that require 24 hour access. Thus another future work is to explore the cache conscious approach to clustering for dynamic clustering. In dynamic clustering, clustering occurs concurrently with database operation. A possible way of doing this is to apply OPCF [15] to C3.

Acknowledgement

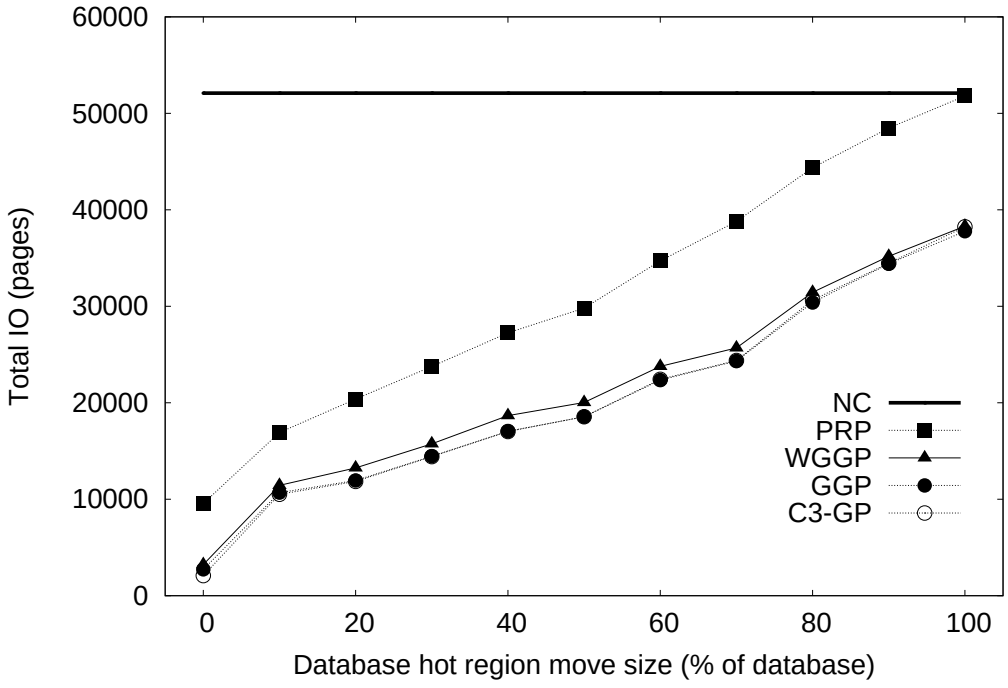
We would like to thank Jerome Darmont for making the sources of VOODB and OCB freely available. These tools have helped tremendously for our simulation study. In addition we would like to thank John Zigman and Stephen Blackburn for their helpful comments and suggestions.

References

1. AILAMAKI, A., DEWITT, D. J., HILL, M. D., AND WOOD, D. A. DBMSs on a modern processor: Where does time go? In *Proceedings of the International Conference on Very Large Databases (VLDB 1999), Edinburgh, Scotland* (September 1999), pp. 266–277.
2. BANERJEE, J., KIM, W., KIM, S. J., AND GARZA, J. F. Clustering a DAG for CAD databases. In *IEEE Transactions on Software Engineering* (November 1988), vol. 14, pp. 1684–1699.
3. BELADY, L. A. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal 5*, 2 (1966).
4. BENZAKEN, V., AND DELOBEL, C. Enhaning performance in a persistent object store: Clustering strategies in o_2. Tech. Rep. 50-90, Altair, August 1990.
5. CAREY, M. J., FRANKLIN, M. J., LIVNY, M., AND SHEKITA, E. J. Data caching tradeoffs in client-server dbms architectures. In *Proceedings of the International conference on the Management of Data (ACM SIGMOD 1991)* (1991), J. Clifford and R. King, Eds., pp. 357–366.
6. CHILIMBI, T. M., DAVIDSON, B., AND LARUS, J. R. Cache-conscious structure definition. In *Proceedings of ACM SIGPLAN conference on programming languages design and implementation* (1999), pp. 13–24.



(a) 1MB buffer size.



(b) 4MB buffer size.

Fig. 6. Comparing how the different static clustering algorithms perform (in terms of total IO (pages)) under different amounts of training skew. Uses the LRU replacement policy.

7. DARMONT, J., PETIT, B., AND SCHNEIDER, M. OCB: A generic benchmark to evaluate the performances of object-oriented database systems. In *Proceedings of the International Conference on Extending Database Technology (EDBT 1998)* (Valencia Spain, March 1998), LNCS Vol. 1377 (Springer), pp. 326–340.
8. DARMONT, J., AND SCHNEIDER, M. VOODB: A generic discrete-event random simulation model to evaluate the performances of OODBs. In *Proceedings of the International Conference on Very Large Databases (VLDB 1999), Edinburgh, Scotland* (September 1999), pp. 254–265.
9. DENNING, P. J. The working set model of program behavior. *Commincations of ACM 11*, 5 (May 1968), 323–333.
10. DREW, P., KING, R., AND HUDSON, S. E. The performance and utility of the cactis implementation algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB 1990)* (Brisbane, Queensland, Australia, 13–16 Aug. 1990), D. McLeod, R. Sacks-Davis, and H.-J. Schek, Eds., Morgan Kaufmann, pp. 135–147.
11. FRANKLIN, M. J., CAREY, M. J., AND LIVNY, M. Local disk caching for client-server database systems. In *Proceedings of the International Conference on Very Large Databases (VLDB 1993)* (1993), R. Agrawal, S. Baker, and D. A. Bell, Eds., pp. 641–655.
12. GERLHOF, C., KEMPER, A., KILGER, C., AND MOERKOTTE, G. Partition-based clustering in object bases: From theory to practice. *Proceedings of the International Conference on Foundations of Data Organisation and Algorithms (FODO 1993)* (1993), 301–316.
13. GRAY, J., AND PUTZOLU, G. R. The 5 minute rule for trading memory for disk accesses and the 10 byte rule for trading memory for cpu time. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1987)* (1987), pp. 395–398.
14. HE, Z., AND MARQUEZ, A. On improving oodbms performance using cache conscious clustering. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA 01)* (2001), pp. 815–825.
15. HE, Z., MARQUEZ, A., AND BLACKBURN, S. Opportunistic prioritised clustering framework (OPCF). In *Proceedings of the International Symposium on Object and Databases* (June 2000), vol. 1944 of *LNCS*, pp. 86–100.
16. HUANG, X., BLACKBURN, S. M., MCKINLEY, K. S., MOSS, E. B., WANG, Z., AND CHENG, P. The garbage collection advantage: Improving program locality. In *Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages and Applications(OOPSLA)* (2004), pp. 69–80.
17. JOHNSON, T., AND SHASHA, D. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the International Conference on Very Large Databases (VLDB 1994)* (1994), pp. 439–450.
18. KNAFLA, N. *Prefetching Techniques for Client/Server, Object-Oriented Database Systems*. PhD thesis, University of Edinburgh, 1999.
19. LEE, D., CHOI, J., KIM, J.-H., NOH, S. H., MIN, S. L., CHO, Y., AND KIM, C. S. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1999)* (1999), pp. 134–143.
20. NICOLA, V. F., DAN, A., AND DIAS, D. M. Analysis of the generalized clock buffer replacement scheme for database transaction processing. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1992)* (1992), pp. 35–46.
21. O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1993) , Washington, D.C.* (1993), P. Buneman and S. Jajodia, Eds., ACM Press, pp. 297–306.
22. ROBINSON, J., AND DEVARAKONDA, N. Data cache management using frequency-based replacement. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1990)* (1990), pp. 134–142.
23. RUBIN, S., BODIK, R., AND CHILIMBI, T. An efficient profile=analysis framework for data-layout optimizations. In *Proceedings of ACM Symposium on the Principles of Programming Languages* (2002), pp. 140–153.
24. 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.
25. TSANGARIS, E.-M. M. *Principles of Static Clustering For Object Oriented Databases*. PhD thesis, University of Wisconsin-Madison, 1992.
26. YUE, P. C., AND WONG, C. K. On the optimality of the probability ranking scheme in storage applications. *JACM 20*, 4 (October 1973), 624–633.