Opportunistic Prioritised Clustering Framework for Improving OODBMS Performance*

Z. He[1] R. Lai[1] A. Marquez[2] S. Blackburn[2]

[1]*Department of Computer Science and Computer Engineering, La Trobe University*
Bundoora, VIC 3086, Australia
{*z.he,r.lai*}*@latrobe.edu.au*
[2]*Department of Computer Science, The Australian National University*
Canberra, ACT 0200, Australia
{*alonso, steveb*}*@cs.anu.edu.au*

Abstract

In object oriented database management systems, clustering has proven to be one of the most effective performance enhancement techniques. Existing clustering algorithms are mainly static, that is re-clustering the object base when the database is off-line. However, this type of re-clustering cannot be used when 24-hour database access is required. In such situations *dynamic clustering* is necessary, since it can recluster the object base while the database is in operation. We find that most existing dynamic clustering algorithms do not address the following important points: the *use of opportunism* to impose the smallest I/O footprint for re-organisation; the *re-use of prior research* on static clustering algorithms; and the *prioritisation of re-clustering* so that the worst clustered pages are re-clustered first. Our main achievement in this paper is to create the Opportunistic Prioritised Clustering Framework (OPCF). The framework allows any static clustering algorithm to be made dynamic. Most importantly it allows the created algorithm to have the properties of I/O opportunism and clustering prioritisation which are missing in most existing dynamic clustering algorithms. We have used OPCF to make the static clustering algorithms "Graph Partitioning" and "Probability Ranking Principle" into dynamic algorithms. In our simulation study we found these algorithms outperformed two existing highly competitive dynamic algorithms in a variety of situations.

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. On the other hand, cheap disks mean object bases 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 object graph navigation is common, I/O may be a more common source of bottleneck.

Ever since the 'early days' of database management systems, clustering has proven to be one of the most effective performance enhancement techniques [12]. Clustering objects in an object oriented database refers to grouping related object onto the same disk page to reduce disk I/O. In addition to reduced I/O, clustering also uses cache space more efficiently by reducing the number of unused objects that occupy the cache.[1] Clustering is effective because the majority of object accesses in an object oriented database are navigational. Consequently, related objects are often accessed consecutively. Periodical re-clustering allows the physical organisation of objects on disk to more closely reflect the prevailing pattern of object access.

The majority of existing clustering algorithms are static [21, 2, 13, 9]. Static clustering algorithms require that re-clustering take place when the database is off-line, thus prohibiting 24 hour database access. In contrast, dynamic clustering algorithms re-cluster the database while database applications are in operation. This makes dynamic clustering algorithms suitable for use in 24 hour databases.

---

* A preliminary version of this paper was presented at *the 1st International Symposium on Objects and Databases 2000*, Sophia Antipolis, France , June, 2000.

[1] Throughout this paper we use the term 'cache' to refer to the in-memory portion of an object base.

In this paper we aim to enhance the performance of OOBDMS by reducing disk I/O. The I/O reduction is accomplished through rearranging the mapping from objects to pages (reclustering). In our problem the OODBMS can not be put off-line during the reclustering process. Hence we develop dynamic clustering algorithms.

In our view, three properties are missing from most existing dynamic clustering algorithms. These properties include:

- The use of opportunism (only recluster pages that are currently in memory) to minimise the I/O footprint for re-organisation; and
- A prioritisation of re-clustering so the worst clustered pages are re-clustered first.
- The re-use of existing work on static clustering algorithms;

In this paper we introduce the Opportunistic Prioritised Clustering Framework (OPCF) which is a transformation framework that converts any static clustering algorithm into a dynamic clustering algorithm that incorporates the first two properties listed above. Next we justify each of the above properties.

Opportunism refers to only reclustering pages that are currently in memory. This simple rule is of critical importance, since the goal of dynamic clustering is to reduce the number of I/Os generated by the database application. So if we generate addition I/O overhead while performing the dynamic clustering then it is very possible that the cost of clustering may out weigh the benefits. By doing clustering opportunistically (only reclustering in memory pages) we can make sure no I/O is generated during the clustering process itself.

Prioritisation refers to choosing the worst clustered page to recluster first. This is very important since at each reorganisation iteration we can only recluster a small portion of the object base (to minimize disruption to the database application), this means we need to be very selective on what we recluster first. Reclustering the worst clustered page first ensures that we get the largest benefit from reclustering as soon as possible.

Despite the great body of work that exists on static clustering[21, 2, 13, 9], there has been little transfer of ideas into the dynamic clustering literature. Existing static clustering algorithms such as the 'graph partitioning' algorithm have been proven to provide best performance for a large variety of situations and datasets. Using OPCF we leverage the strengths of these techniques to produce new dynamic clustering algorithms that outperform existing dynamic clustering algorithms. To this end we use OPCF to produce two new dynamic clustering algorithms by incorporating two existing and yet vastly contrasting types of static clustering algorithms into our dynamic clustering framework. These are the 'graph partitioning' and the 'probability ranking principle' (PRP) algorithms. We show that by using OPCF these two existing static clustering algorithms outperform existing highly competitive dynamic clustering algorithms, DSTC[3] and DRO [5] in a variety of situations.

The main contribution of this paper is the development of the OPCF framework which transforms static clustering algorithm into dynamic algorithms. The dynamic clustering algorithms that are produced incorporates the desirable properties of opportunism and prioritization.

2 Related Work

We divide this section into two sections. First an overview of existing dynamic clustering algorithms. Second a detailed description of existing clustering algorithms that have been used in our simulation study.

2.1 Overview of Existing Dynamic Clustering Algorithms

The re-organisation phase of dynamic clustering can incur significant overhead. Two of the key overheads are increased write contention,[2] and I/O.

To reduce write contention, most dynamic clustering algorithms are designed to be incremental and thus limit the scope of re-organisation. However, *DROD* [24] is the only algorithm that we are aware of that limits the scope of re-organisation so that only in-memory objects are re-clustered. Wietrzyk et. al.[24] accomplish this by calculating a new placement when the object graph is modified, either by a link (reference) modification or object insertion. The algorithm then re-clusters the objects that are effected by the modification or insertion. Once the new placement is determined, only the objects in memory are re-organised and the remaining objects are only re-arranged as they are loaded into memory. However, the statistical data required by DROD has *global* scope (statistics about any object in the store may be needed). In contrast, OPCF has *local* scope in terms of statistical requirements (only statistics of in-memory objects are required).

[2] Note that in an optimistic system this translates to transaction aborts.

The incremental nature of dynamic clustering requires that only a small portion of the entire database be re-clustered at each iteration. However, the choice as to which portion to re-cluster is where many existing algorithms differ. William et. al.[19] suggest targeting the portion that was accessed after the previous re-organisation. However, this may involve a very large portion of the database if the re-clustering is not triggered frequently. Wietrzyk et. al. in [24] re-cluster affected objects as soon as an object graph modification occurs. They use a threshold mechanism[3] to determine when re-clustering is worthwhile. However, this approach may still be too disruptive. An example of when its disruptiveness is likely to be felt is when the system is in peak usage and frequent object graph modifications are occurring. In such a scenario the object graph would be continuously re-clustered during peak database usage. The algorithm thus lacks a means of controlling when the re-clustering takes place. In contrast, the dynamic algorithms developed with OPCF can be easily made adaptive to changing system loads. This is due to the fact that re-clustering can be triggered by an asynchronous dynamic load-balancing thread rather than an object graph modification.

The dynamic clustering algorithms *StatClust* [11] and *DRO* [5] identify and re-cluster all pages containing objects that have a quality of clustering lower than a threshold amount. If the number of poorly clustered pages (pages below clustering quality threshold) is very high, then these approaches would re-cluster a large number of pages within the same re-organisation iteration. In contrast, OPCF ranks pages in terms of quality of clustering and then only re-clusters a *bounded* number of the worst clustered pages. This allows OPCF to bound the number of pages involved in each re-organisation iteration to a user defined number of pages (the user can decide the maximum amount of interruption he or she tolerates).

The DSTC dynamic clustering algorithm identifies and re-clusters all pages that can be improved by clustering [3]. Therefore, even if there is only very small possible improvement the page will be re-clustered. This leads to over vigorous re-clustering which produces poor overall performance. However, DSTC does take care to limit the number of pages involved in each re-organisation iteration by breaking the re-organisation workload into re-clustering units and only re-organisating one unit in each iteration.

A large body of work exists on static clustering algorithms [21, 2, 13, 9]. However, only relatively few static algorithms have been transformed into dynamic algorithms. William et. al. [19] combined the existing static clustering algorithms, Cactis [16] and DAG [2], to create a new dynamic clustering algorithm. However Cactis and DAG are only sequence-based clustering algorithms which have been found to be inferior when compared to graph partitioning algorithms [21]. Wietrzyk at. el. in [24] develop a new dynamic graph partitioning clustering algorithm. However, they do not compare their dynamic graph partitioning algorithm with any existing dynamic clustering algorithm. In this paper, two existing static clustering algorithms are transformed into dynamic clustering algorithms using OPCF and compared to two existing dynamic clustering algorithms, DSTC [3] and DRO [5].

2.2 Detailed Description of Two Existing Dynamic Clustering Algorithms

In order to give readers better understanding of our simulation study, in this section we describe the two existing dynamic clustering algorithms used in the simulations. Neither of the algorithms are opportunistic and both offer only limited incrementality and prioritisation.

Dynamic Statistical and Tunable Clustering (DSTC) DTSC is an existing dynamic clustering algorithm [3] designed to achieve dynamicity without adding high overhead or an excessive volume of statistics.

The algorithm is comprised of five phases:

- *Observation Phase*: In order to minimise disruptiveness of statistics collection, DSTC only collects statistics at predefined observation periods and the information is stored in a transient observation matrix.
- *Selection Phase*: In order to reduce the volume of statistics stored, at the selection phase the transient observation matrix is scanned and only significant statistics are saved.
- *Consolidation Phase*: The results of the selection phase are combined with statistics gathered in previous observation phases and saved in a persistent consolidated matrix.
- *Dynamic Cluster Re-organisation*: Using the information in the updated consolidated matrix, new clusters are discovered or existing ones are updated. In order to achieve incrementality, the re-organisation work is broken up into small fragments called clustering units.
- *Physical Clustering Organisation*: The clustering units are finally applied to the database in an incremental way (that is, one clustering unit at a time). This phase is triggered when the system is idle.

[3] Based on the heuristic more frequently accessed objects that are clustered badly should be re-clustered earlier.

DSTC uses sequence tension information to model access dependencies between objects. DSTC is not an *opportunistic* clustering algorithm since its scope of re-organisation can be objects that are currently residing on disk. DSTC exhibits a small degree of prioritisation since it breaks the database into objects that can be improved from clustering (worse clustered) and ones that cannot (better clustered). Even if an object can only potentially get a very small clustering improvement, it is re-clustered. This approach generates a lot of clustering overhead which often cannot be justified by the relatively small clustering quality improvements.

Detection & Re-clustering of Objects (DRO) Learning from the experiences of DSTC and StatClust [11], DRO [5] is designed to produce less clustering I/O overhead and use less statistics. In order to limit statistics collection overhead, DRO only uses *object frequency* (heat) and *page usage rate* information. In contrast, DSTC keeps sequence tension information which is much more costly.

DRO is a *page* grained dynamic clustering algorithm. Therefore it re-clusters all objects in pages that are selected for re-clustering.

The DRO clustering algorithm is comprised of 4 steps:

1. *Determination of Objects to Cluster:* In this step various thresholds are used to limit the pages involved in re-clustering to only those pages that are most in need of re-clustering. For a page to require re-clustering the following conditions need to be met: the fraction of unused objects must be lower than the $MinUR$ threshold; and the amount of I/O that the page generated must be greater than the $MinLT$ threshold. To proceed to step 2 the ratio between the number of pages needing re-clustering and number of pages actually used must be greater than a threshold rate $(PCRate)$.
2. *Clustering Setup:* This step takes the objects in the pages in need of re-clustering and then generates a new placement order of objects on disk. The placement algorithm runs as follows: objects of similar heat and with structural links are placed closer together in the new placement order. Then the new placement order is compared against the old and the algorithm only proceeds to the next step if there is enough difference $(MAXRR)$ between the two placement orders.
3. *Physical Object Clustering:* This operation physically clusters objects identified in the previous steps, but must also re-organise the database in order to free space made available by the deletion or movement of objects.
4. *Statistics Update:* This step resets clustering statistics. Depending on the update indicator $(SUInd)$ parameter, all pages or just the pages involved in the re-clustering are reset.

DRO is not opportunistic since disk resident pages can be involved in re-clustering. DRO has only limited incrementality since at each iteration it re-organises all pages that are clustered worse than a threshold amount. If the number of pages in need of clustering is high, DRO will become less incremental. In contrast, our approach ranks all pages in terms of quality of clustering and then re-clusters only a user-defined number of the worst clustered pages. Our approach allows the user to limit the amount of re-clustering that he or she is willing to accept in each re-organisation iteration, whereas DRO has no such limit. DRO prioritises clustering by breaking up the database into pages that need re-clustering (according to thresholds) and pages that do not. This method of prioritisation has the benefit that when database clustering quality is very low, fewer pages are re-clustered and the reverse when clustering quality is high. This more flexible behaviour of DRO when compared to OPCF is at the cost of good incrementality (the ability to ensure only a bounded portion of database is involved in each re-organisation iteration).

3 Assumptions, Definitions

3.1 Assumptions

The work in this paper makes the following assumptions:

1. The object to page mapping can be changed from one consistent state to another without ever exposing client threads to inconsistent mappings.
2. All objects are smaller than one page in size. Since large objects[4] do not benefit from clustering, we choose to focus our study on objects smaller than a page. 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 the dynamic clustering algorithm ignores them.
3. The patterns of object access after and before each re-organising iteration bares some degree of similarity.

[4] Objects larger than one page in size.

3.2 Definitions

Tsangaris at. el. [22, 23] proposed two metrics for measuring the quality of an object clustering—working set size and long term expansion factor.

Working set size (WSS(M)) [22] is a metric for clustering quality that is independent of the buffer replacement algorithm (algorithm used to decide which page to evict from memory when the buffer is full). This means it measures how well related objects are grouped together irrespective of the algorithm used to decide which page to evict when the memory is full. WSS(M) is evaluated by taking M page requests, eliminating duplicates and computing the the resulting number of pages. Therefore, the larger the number of resulting pages, the worse the clustering. Since the M page requests are spread across a larger number of pages. A clustering algorithm that achieves a lower value for this metric (better quality clustering) will perform well on workloads that traverse a small portion of the database starting with a cold cache (when the cache is empty). The reason is a lower value for this metric means less pages will need to be loaded if there is currently no pages in memory.

Long term expansion factor EF_∞ [23] is an indicator of the long term performance of an object clustering algorithm when the cache size is large. EF_∞ is the ratio of the number of pages the all active objects occupy compared to the minimum number of pages to store all active objects. The most ideal EF_∞ value would be one, since that would be the situation where all pages accessed would contain active objects only. In general the smaller the value of EF_∞ is the better the clustering algorithm performs since it would require a smaller cache size to store all active objects in memory.

It is important to remember that these metrics are independent of buffer replacement algorithms and thus do not accurately predict algorithm performance. They are included in this paper to serve as a tool for discussing the relative merits of existing static clustering algorithms.

4 Opportunistic Prioritised Clustering Framework (OPCF)

In this section we outline in detail the main contribution of this paper, the Opportunistic Prioritised Clustering Framework (OPCF). OPCF transforms static clustering algorithms into dynamic algorithms and provides them with the attributes of *opportunism*, *incrementality*, and *prioritisation*. We begin by describing how OPCF achieves these attributes. We then define the steps of the OPCF framework.

4.1 Properties of OPCF

OPCF introduces the *opportunism* property to minimise read I/O overheads caused by the dynamic clustering thread. OPCF achieves opportunism by restricting clustering to *in-memory* pages only.

OPCF limits the disruption caused by the dynamic clustering process by *incrementally* re-organising the database. OPCF achieves incrementality by placing a fixed bound on the number of pages re-clustered in each re-clustering iteration and then allowing users to control the time between successive re-clustering iterations.

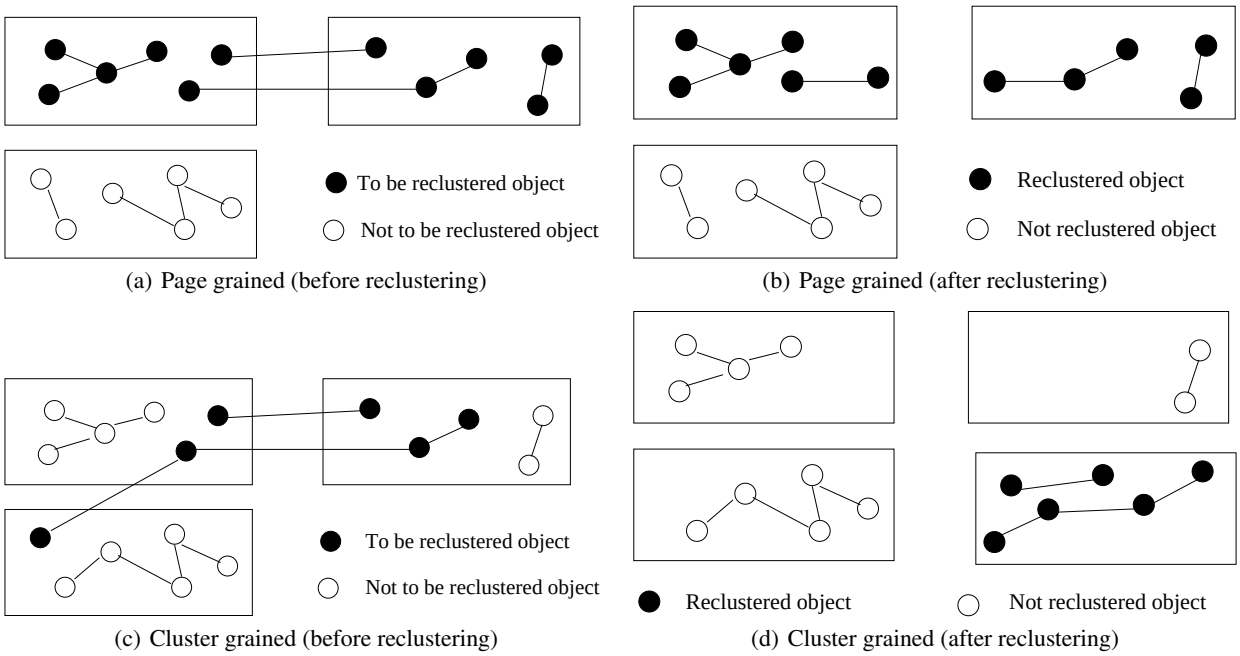
Incrementality specifies that re-organisation should be partitioned and only one portion of the database should be re-organised in each iteration. OPCF incorporates the desirable property of *Prioritisation*. Prioritisation specifies that the worst clustered portion of the database should be targeted for re-organisation in each iteration. Prioritisation aims to achieve large reductions of I/O incurred by the database application while incurring only small amounts of I/O and CPU cost for the clustering process. OPCF performs prioritisation by ranking pages in terms of quality of clustering and then limiting re-organisation to a user-specified set of the worst clustered pages.

4.2 Framework Definition

OPCF works at the *page* grain, instead of *cluster* grain. This means *all* objects in pages selected for re-clustering are re-clustered. In contrast, *cluster* grain algorithms like DSTC [3] remove *selected* objects that are determined to need re-clustering from existing pages and place them into *new* pages. The cluster grained approach as two main drawbacks: 1) objects in the cluster which are in disk will need to be loaded into memory in order to be reclustered, 2) new pages will need to be created to places objects of the cluster, which increases the number of pages the object base is spread across.

Figure 1 shows an example that illustrates the different between page and cluster grained reclustering before and after reclustering. In the case of page grained reclustering either all the objects in the page are reclustered or none are. Whereas in the cluster grained case only the objects most in need are reclustered and then placed in new pages.

In order to create OPCF algorithms, a series of steps must be applied to the static clustering algorithm that is to be transformed into a dynamic algorithm. These steps are shown below.



(a) Page grained (before reclustering) (b) Page grained (after reclustering)

(c) Cluster grained (before reclustering) (d) Cluster grained (after reclustering)

Fig. 1. An example illustrating the difference between page and cluster grained clustering. The example shows both before and after reclustering.

- *Define Incremental Re-organisation Algorithm*: In this step the existing static clustering algorithm is modified to work in an incremental way. That is, at each iteration of re-organisation the algorithm must recluster a portion of the object base.
- *Define Clustering Badness Metric*: OPCF prioritises re-clustering by re-clustering the worst clustered pages first. Therefore in this step a metric that measures the quality of clustering at the page grain is defined. We term this the *clustering badness metric*. The way in which clustering badness is to be defined for a particular static clustering algorithm depends on the goal of the clustering algorithm.

  For instance, the PRP clustering algorithm has the goal of grouping hot[5] objects together and therefore it may have a clustering badness metric that includes a measure of the concentration of cold objects in pages that contain hot objects.

  At each clustering analysis (refers to period when clustering badness is computed) iteration , a user-defined number of pages (NPA) have their clustering badness calculated. Once a page's clustering badness is calculated, it is compared against a user-defined clustering badness threshold (CBT). If the page has a higher clustering badness value than the threshold, then the page is placed in a priority queue sorted on clustering badness. At each re-organisation iteration a page is removed from the top of the priority queue and used to determine the scope of re-organisation for that re-organisation iteration. A user-defined number (NRI) of re-organisation iterations are performed at the end of each clustering analysis iteration.
- *Define Scope of Re-organisation*: To limit the work done in each re-organisation iteration of the dynamic clustering algorithm, a limited number of pages surrounding the worst clustered pages is chosen to be reclustered with the worst clustered page. We call these pages the scope of re-organisation. In this step the scope of re-organisation is chosen. The scope of re-organisation should be chosen in such a way that re-organisation of those pages will produce the maximum amount of improvement in clustering quality while preserving the property of incrementality.

  The way the scope of re-organisation is chosen dictates whether the clustering algorithm is opportunistic or non-opportunistic. To achieve opportunism, only *in-memory* pages are included in the scope of re-organisation.
- *Define Cluster Placement Policy*: Because OPCF works at a page rather than cluster grain, the initial stages of each re-organisation iteration target a limited number of pages and so will, in general, identify multiple clusters, some of which may be small. The existence of clusters which are smaller than a page size raises the important

[5] By hot objects we mean objects accessed more frequently.

issue of how best to pack clusters into pages. We call this the cluster placement policy. In this step a cluster place policy is defined.

In OPCF cluster analysis is triggered simply when a user-specified number of objects (N) has been accessed. This is similar to the technique used in DSTC [3]. However, any other triggering method can be used, including triggering via an asynchronous thread (eg. for load balancing reasons).

5 Two Example algorithms generated using OPCF

In this section we present two dynamic clustering algorithms generated using OPCF. We first describe the static clustering algorithms from which our dynamic clustering algorithms are derived. Next we describe in detail how OPCF is used to transform the static clustering algorithms into dynamic algorithms.

5.1 Probability Ranking Principle (PRP)

The static probability ranking principle (PRP) algorithm [21] is the simplest sequence-based clustering algorithm. Sequence-based clustering [2, 9, 21] algorithms have two phases: *presort*; and *traversal*. In the *presort* phase objects are sorted and placed in a sorted list. Some examples of sorting order are: by class; by decreasing heat (where 'heat' is simply a measure of access frequency), etc. During the *traversal* phase the clustering graph[6] is traversed according to a traversal method specified by the clustering algorithm. The roots of the traversals are selected in sorted order from the sorted list. This process produces a linear sequence of objects which are then mapped onto pages. In static PRP, the objects are presorted according to decreasing heat. Then the objects are just placed into pages in this presorted order. This surprisingly simple algorithm yields near optimal long term expansion factor.

Figure 2 shows an example illustrating how the PRP clustering algorithm works. The figure shows both before and after PRP clustering.

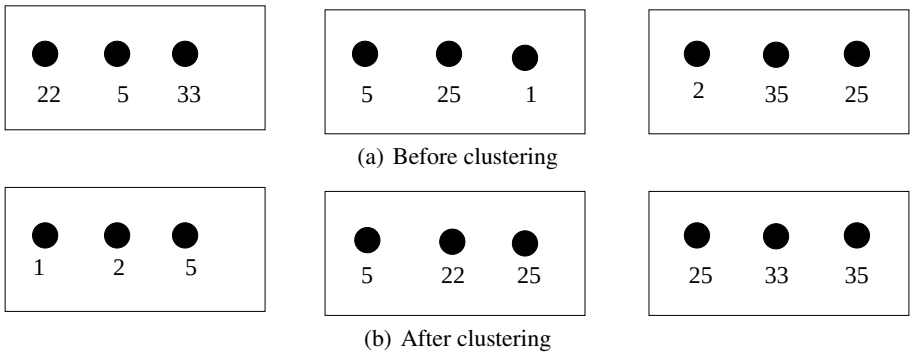


Fig. 2. An example illustrating the PRP clustering algorithm. The number below each object shows the heat of the object.

The reason that PRP achieves a near optimal expansion factor is that it groups together those objects that constitute the active portion of the database. Therefore, when the size of the active portion of the database is small relative to the available cache size and the steady state (after the cache has been warmed up) performance of the database is of interest, this algorithm yields a near optimal solution. However, when a small traversal is conducted on a cold cache, PRP tends to perform poorly for working set size, since it does not take object relationships into consideration [21].

The simplicity of the PRP algorithm (minimal statistical requirements and low time complexity) makes it particularly suitable for dynamic clustering. PRP uses only heat statistics. PRP's time complexity is determined by the sorting of objects in terms of heat and thus has a time complexity of $O(n \log n)$, where n is the number of objects in the database. However, to our knowledge, no dynamic version of PRP has been suggested before in the literature.

[6] Where nodes represent objects and edges represent object references. The edge weights represent the frequency by which the object reference is traversed.

Dynamic Probability Ranking Principle In this section we describe the application of OPCF to the PRP clustering algorithm to transform it into a dynamic clustering algorithm. The reason we have selected PRP is that it has been proven to provide near optimal expansion factor. This means it is very efficient at packing the objects used in the steady state in the smallest number of pages possible. Thus it is particularly suitable to situations where the number of objects used in steady state all fits into memory.

OPCF_PRP(*LP*: list of pages)

1. Compute the clustering badness metric (see Equation 5.1) for each page in *LP*.
2. Let *WP* be the page in *LP* with largest clustering badness value.
3. Let the scope of re-organization list *SRL* = *WP* + two pages in *LP* closest to *WP* on disk.
4. Place all objects in *SRL* in list *L*.
5. Sort objects in *L* in descending order according to heat.
6. Place objects in *L* in sorted order across the pages in *SRL*.

Fig. 3. The dynamic PRP clustering algorithm produced using OPCF.

Figure 3 shows the dynamic PRP clustering algorithm produced using OPCF. When *LP* is restricted to all pages currently in memory then the algorithm is run opportunistically. The algorithm is run periodically to incrementally recluster the object base. The following describes the algorithm in terms of the steps of the OPCF framework:

- *Incremental Re-organisation Algorithm*: In order to make PRP work in an incremental fashion, a logical ordering based on heat is placed on the pages of the store. The clustering algorithm incrementally re-arranges the objects so as to slowly migrate cold objects to cold pages and hot objects to hot pages.
  At each re-organisation iteration, the algorithm reorders the set of objects that lie within the pages targeted for that iteration according to heat order, the hottest objects moving to the hottest page, the coldest to the coldest page, etc.
- *Clustering Badness Metric*: The goal of the PRP clustering algorithm is to map the active portion of the database into as few pages as possible. It accomplishes this by migrating hot objects towards one portion of the store while migrating cold objects towards the other direction. In order to achieve this objective, we have defined a clustering metric which says a page is worse clustered if it contains hot objects and a larger amount of waste. We define waste to mean space consumed by cold objects. The intuition behind this definition of clustering badness is that pages which contain hot objects but also a lot of waste are both very likely to be in the cache and also wasting a lot of cache space, and thus unnecessarily displacing other hot objects.
  The definition of clustering badness of page p is as follows:

$$CB(p) = \left(\sum_{i \in p} heat_i \right) \times \left(\sum_{i \in p} (size_i / heat_i) \right) \quad (1)$$

  In the above equation i refers to the i^{th} object in page p. $heat_i$ refers to the heat of the i^{th} object. As mentioned earlier heat of an object equals the access frequency of the object. $size_i$ refers to the size of the i^{th} object. The second term in the equation is a measure of the waste in the page. Therefore a larger and colder object in a page will contribute more waste.
- *Scope of Re-organisation*: The scope of each re-organisation is defined as three pages which are adjacent in heat-order, where the middle page is the target page for that iteration and the target page is chosen to be the page which is currently worst clustered. When opportunism is used, the two *in-memory* pages closest to the target are selected (as the adjacent pages may be on disk). See figure 4 for an example.
  This definition of scope of re-organisation gives the clustering algorithm a high degree of incrementality. In addition, this gives the clustering algorithm an opportunity to improve the quality of clustering by placing the colder objects in the logically colder page and hotter objects in the logically hotter page.
- *Cluster Placement Policy*: Since PRP does not produce clusters of objects, it does not have a cluster placement policy.

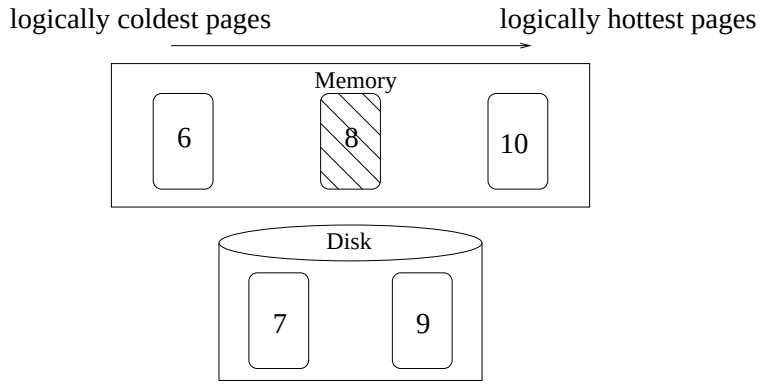


Fig. 4. In this example the currently worst clustered page is 8, so the scope of re-organisation for opportunistic dynamic PRP is pages 6, 8 and 10 (where page numbers reflect heat-order). If opportunism is not used, the scope would be pages 7, 8 and 9.

The time complexity of this algorithm per re-organisation iteration is $O(n_s \ log \ n_s)$, where n_s is the number of objects within the scope of re-organisation. The time complexity is determined by the sorting of objects within the scope of re-organisation.

5.2 Greedy Graph Partitioning

Partition-based 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 vertices and edges of the graph are labeled with weights. Vertex weights represent object size. Edge weights represent either the frequency of *structural reference* traversal or the transition probabilities from the Simple Markov Chain model. We will term the former *structural tension* and the later *sequence tension*.

There are two types of partition-based static clustering algorithms: *iterative improvement* and *constructive partitioning*. Iterative improvement algorithms such as the Kernighan-Lin Heuristic (KL) [17], iteratively improve partitions by swapping objects between partitions in an attempt to satisfy the min-cut criteria. Since KL swaps objects between partitions it requires objects to be relatively uniform in size which makes it inappropriate for many real world OODBs. Constructive algorithms such as greedy graph partitioning (GGP)[13] attempt 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).

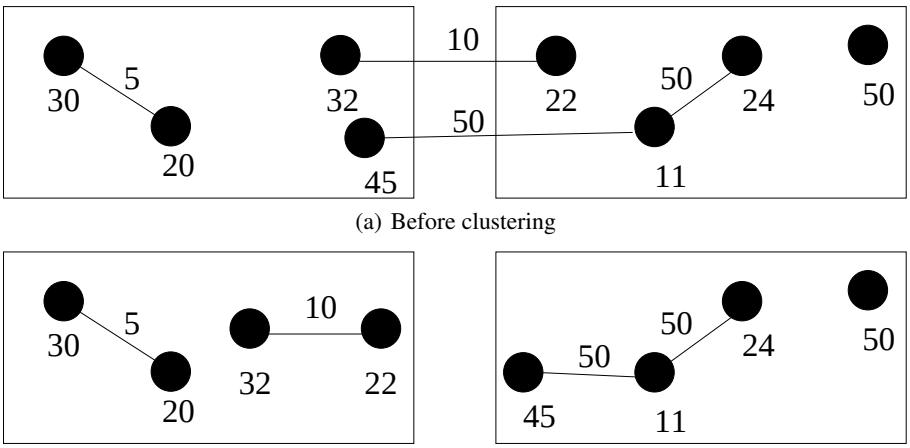
The study carried out by Tsangaris and Naughton [21] indicates that graph partitioning algorithms perform best for both the working set size metric and long term expansion factor metric. However, they are generally more expensive in terms of CPU usage and statistic collection (tension statistics) than sequence-based algorithms. The time complexity of the KL graph partitioning algorithm is $O(n^{2.4})$ and $O(e \ log \ e)$ for GGP, where n is the number of vertices and e is the number of edges.

Figure 5 shows an example of a clustering graph before and after GGP clustering. The number below each object shows the size of the object and the number above each edge show the frequency by which the edge is traversed. GGP performs the clustering by satisfying the min-cut criteria. In the Figure 5 (a) (before the GGP algorithm is run) the total clustering graph edge weight cut is 60. After GGP clustering (Figure 5 (b)) the total weight of graph edges cut equals zero. Comparing Figure 5 (a) and Figure 5 (b) it is clear that after GGP clustering there much less navigations crossing page boundaries.

Dynamic Graph Partitioning This section outlines how we use OPCF to transform static graph partitioning algorithms into dynamic algorithms. The reason we have chosen static graph partitioning is that it has been shown to be the best performing static clustering algorithm in a variety of datasets and situations[21].

Figure 6 shows the dynamic GGP clustering algorithm produced using OPCF. When *LP* is restricted to all pages currently in memory then the algorithm is run opportunistically. The algorithm is run periodically to incrementally recluster the object base. The following describes the algorithm in terms of the steps of the OPCF framework:

– *Incremental Re-organisation Algorithm*: At each re-organisation iteration, the graph partitioning algorithm is applied to the pages in the scope of re-organisation as if these pages represent the entire database.



(a) Before clustering

(b) After clustering

Fig. 5. An example illustrating the GGP clustering algorithm.

OPCF_GGP(*LP*: list of pages, ETT external tension threshold)

1. Compute the clustering badness metric (see Equation 2) for each page in *LP*.
2. Let WP be the page in *LP* with largest clustering badness value.
3. Let the scope of re-organization list $SRL = WP$.
4. For each page p in *LP*
5. if (extern tension of p in respect to $WP >$ external tension threshold ETT)
6. Add p into *SRL*.
7. Run the static GGP algorithm to recluster the pages in *SRL*.

Fig. 6. The dynamic GGP clustering algorithm produced using OPCF.

– *Clustering Badness Metric*: The static graph partitioning algorithms attempt to satisfy the min-cut criteria. This means that they minimise the sum of edge weights that cross page boundaries. In order to include this criteria into our clustering badness metric we have included external tension in the metric. We define external tension as the sum of edge weights of the clustering graph which cross page boundaries. A page with higher external tension is worse clustered. In addition, heat is included in the metric to give priority for re-organising hotter pages. Below is a definition of clustering badness for page p:

$$CB(p) = (\sum_{i \in p} heat_i) \times (\sum_{i \in p} external\ tension_i) \quad (2)$$

The calculation of external tension differs between the opportunistic version of the dynamic graph partitioning algorithm and the non-opportunistic version. In the opportunistic version, the external tension is calculated from only the weights of edges that cross the boundary of the page under consideration to other *in-memory* pages. In contrast, the non-opportunistic algorithm counts edge weights that crosses page boundaries onto *disk* pages.
– *Scope of Re-organisation*: The scope of re-organisation is the worst clustered page and its related pages. A page is only considered related if it occupies an external tension threshold (ETT) fraction of the worst clustered page's external tension. This reduces the scope of re-organisation to the pages that will benefit the most from the re-organisation. ETT acts as a means of trading off clustering quality with clustering overhead. If the dynamic clustering algorithm is to be run opportunistically then only *in-memory related* pages are in the scope of re-organisation. See figure 7 for an example.
– *Cluster Placement Policy*: For this application of OPCF, we have chosen to place clusters into pages in order of heat. The reason for this choice is that cold clusters will be placed away from hot clusters and thus pages containing hot clusters which are more likely to be in memory will have less wasted space occupied by cold clusters. This is similar to the goal of PRP (see section 5.1).

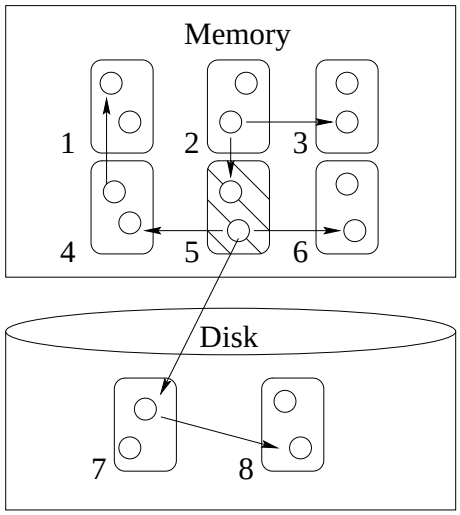


Fig. 7. In this example the worst clustered page is 5 and the scope of re-organisation for opportunistic dynamic graph partitioning are pages 2, 4, 5 and 6. The scope of re-organisation for non-opportunistic dynamic graph partitioning are pages 2, 4, 5, 6 and 7.

The particular graph partitioning algorithm implemented for the results section of this paper is the greedy graph partitioning (GGP) algorithm [13]. However, the above methodology can be applied to any static graph partitioning clustering algorithm. GGP first places all objects in a separate partition and then iterates through a list of edges in descending edge weight. If the two objects on the ends of the currently selected edge are in different partitions and the total size of the two partitions is smaller than a page, then the partitions are joined. GGP uses sequence tension to model access dependencies between objects. The time complexity of our dynamic GGP algorithm is $O(e_s\ log\ e_s + p_s\ log\ p_s)$, where e_s is the number of edges in the scope of re-organisation and p_s is the average number of initial partitions generated after the first three steps. The time complexity is determined by the sorting of clustering graph edge weights and the sorting of the initial partitions generated according to heat. In most cases e_s is much larger than p_s and therefore in most cases the time complexity is $O(e_s\ log\ e_s)$.

6 Simulation Setup

In this section we describe simulations designed to compare the performance of algorithms produced using OPCF with two existing state of the art dynamic clustering algorithms, DSTC and DRO. We have chosen DSTC and DRO because they are two of the latest dynamic clustering algorithms. They are built based on the lessons learnt from many older dynamic clustering algorithms.

6.1 Simulation Environment

The simulations are conducted using the virtual object oriented database simulator, VOODB [7]. VOODB is based on a generic discrete-event simulation framework. Its purpose is to allow performance evaluations of OODBs in general, and optimisation methods such as clustering in particular. VOODB simulates all of the traditional OODBMS components such as: the transaction manager, object manager, buffer manager, and I/O subsystem. The correctness of VOODB has been validated for two real-world OODBs, O_2 [8] and Texas [20].

We use simulation for two reasons. First, it allows rapid development and testing of a large number of dynamic clustering algorithms (all existing dynamic clustering papers compared at most two algorithms). Second, it is relatively easy to simulate accurately read, write and clustering I/O (the dominating metrics that determine the performance of dynamic clustering algorithms).

It is important to note that in terms of write I/O, our simulation study shows close to worst case scenario. The reason is that we do not simulate a flush thread, instead we only flush dirty pages during dirty page eviction time. Flush threads allow dirty pages to be flushed to disk in the background. It is beyond the scope of this paper to explore the effects that different flushing policies have on dynamic clustering algorithm performance. However, in general, all policies will show better performance when fewer pages are dirtied. The negative effect of dirtied pages is reflected in our simulator by the flushing of dirty pages during eviction.

VOODB is very user tunable, offering a number of system parameters such as: system class (distribution configuration), page size, buffer size, buffer replacement strategy, etc. The parameters relevant to our study along with the values used are depicted in table 1. The 'centralised' system class refers to the stand-alone system configuration, in which the clients and server both reside on the same machine. Optimised sequential placement refers to compact placement (every page has a low percentage of empty space), which is mostly placed in creation order.

Parameter Description	**Value**
System class.	Centralised
Disk page size.	4096 bytes
Buffer size.	varies
Buffer replacement strategy.	LRU-1
Pre-fetching strategy.	None
Object initial placement.	optimised sequential

Table 1. VOODB parameters used.

6.2 Performance Evaluation Environment

We evaluate the performance of the dynamic clustering algorithms in two sets of simulations. First, we use the object clustering benchmark (OCB) [6] to compare the performance of the dynamic clustering algorithms under stationary access pattern situations. Second, the dynamic object evaluation framework (DOEF) [15] is used to test the effects of changes in access patterns.

In this paper we use traces generated from synthetic benchmarks instead of real world applications. The main reason is synthetic benchmarks allow us to change various settings and thus generate multiple traces with know characteristics. This allows algorithms to be tuned or compared using multiple styles of access pattern change and thus can give more insights into why algorithms perform in a particular way. In addition, real application traces are very difficult to obtain in the OODBMS field.

settings and thus generate multiple traces with know characteristics. This

However, it has been noted in [25] that the synthetic benchmarks may not produce traces that are indicative of real world applications. The work of [25] has been done in the context of memory allocation in programming languages.

No similar study has been done in OODBMSs. We believe this is because of the difficulties in obtaining real world OODB application traces.

Stationary Access Pattern Evaluation Environment In this paper the OCB benchmark [6] is chosen as the tool used for evaluating buffer management techniques under stationary access patterns. Here we provide a brief description of OCB. In addition, we describe the OCB parameter settings that are used for the simulations in this paper.

The OCB benchmark was initially designed for benchmarking clustering algorithms but its rich schema and realistic workloads make it particularly suitable for benchmarking prefetching algorithms too. 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 reported in this paper, a total of 100, 000 objects are generated. The objects varied in size from 50 to 1600 bytes and the average object size is 232 bytes. The total database size is 23 MB. Although this is a small database size we also use small buffers (1MB and 4MB) to keep the database to buffer size ratio large. Since we are interested in the caching behaviour of the system, the database to buffer size ratio is a more important parameter than database size alone. The OCB database parameters used are shown in table 2 (a).

The OCB workload used in this study included simple, hierarchical and stochastic traversals [6]. The simple traversal performs a depth first search starting from a randomly selected root object. The hierarchical traversal picks a random root object and a random reference type and then always follows the same reference type up to a pre-specified depth. The stochastic traversal selects the next link to cross at random. At each step, the probability of following reference number N is $p(N) = \frac{1}{2^N}$. Stochastic traversals approach Markov chains, which are known to simulate real query access patterns well [23]. Each transaction involved the execution of one of the three traversals chosen randomly. The OCB workload parameters used are shown in table 2 (b).

(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.	100000
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) OCB workload parameters

Parameter Description	**Value**
Simple traversal depth.	2
Hierarchy traversal depth.	4
Stochastic traversal depth.	4
Transaction root selection distribution.	Hot/Cold
Simple traversal selection probability.	0.3
Hierarchical traversal selection probability.	0.35
Stochastic traversal selection probability.	0.35
Number of transactions.	100000

Table 2. Parameters used for OCB.

We introduce skew into the way traversal roots are selected. Roots are partitioned into hot and cold regions. In all simulations the hot region is set to 3% of the size of database and has an 80% probability of access.[7] These settings are chosen to represent typical database application behaviour. Gray at. el. [14] cites statistics from a real videotext application in which 3% of the records got 80% of the references. Carey at. el. [4] use a hot region size of 4% with a 80% probability of being referenced in the HOTCOLD workload used to measure data caching trade-offs in client/server OODBMSs. Franklin at. el. [10] use a hot region size of 2% with a 80% probability of being referenced in the HOTCOLD workload used to measure the effects of local disk caching for client/server OODBMSs.

Dynamic Access Pattern Evaluation Environment In this section we describe the dynamic object evaluation framework (DOEF) [15]. We use DOEF to compare the dynamic clustering algorithms' ability to cope with changing access patterns. DOEF is a synthetic benchmark that models changes in application access pattern behaviour. DOEF accomplishes access pattern change by defining configurable styles of change. DOEF is built on top of OCB, using both the database built from the rich schema of OCB and the operations offered by OCB. Access pattern change is accom-

[7] That is, there is a 80% probability that the root of a traversal is from the hot region.

plished by varying the way traversal roots of OCB are selected. DOEF divides traversal roots[8] into partitions called 'H-regions'. All traversal roots within the same H-region have the same probability of selection, however, traversal roots between different H-regions can have a different probability of selection. Access pattern change is specified by varying the 'heat' of H-regions according to access pattern change protocols.

Two different DOEF protocols of change are used in this paper. First, the *moving window of change* protocol is used to simulate sudden changes in access patterns. This style of change is accomplished by moving a window through the database. The objects in the window have a much higher probability of being chosen as a traversal root when compared to the remainder of the database. This is done by partitioning the database into *N* H-regions of equal size. Then one H-region is first initialised as the hot region, and the remaining H-regions are initialised as cold regions. After a certain number of root selections, a different H-region becomes the hot region.

Second, the *gradual moving window of change* protocol is used to simulate a milder style of access pattern change. This protocol differs from the previous one in that the hot region cools down gradually instead of suddenly. The cold regions also heat up gradually as the window is moved onto them. This tests the dynamic clustering algorithm's ability to adapt to a more moderate style of change.

In the DOEF simulations we use the same OCB database parameters as those shown in table 2 (a). However, we use different workload parameters. The operation we have used for all of the simulations is the simple, depth-first traversal with traversal depth 2. The simple traversal is chosen since it is the only traversal that always accesses the same set of objects given a particular root. This establishes a direct relationship between varying root selection and changes in access pattern. Each simulation involved executing 10,000 transactions.

The DOEF parameters used in this paper are shown in table 3. The 'H-region size' setting of 0.003 (remember this is the database population from which the traversal *root* is selected) creates a hot region about 3% the size of the database (each traversal touches approximately 10 objects). This was verified from statistical analysis of the trace generated. The 'highest H-region access probability' setting of 0.8 and 'lowest H-region access probability setting' of 0.0006, produces a hot region with 80% probability of reference and the remaining cold regions with a combined reference probability of 20%. The 'probability increment size' parameter is used by the gradual moving window of change protocol to specify the amount by which the H-regions change heat at each change iteration.

Parameter Description	**Value**
H-region size.	0.003
Highest H-region access probability.	0.80
Lowest H-region access probability.	0.0006
Probability increment size.	0.02
Object assignment method.	random object assignment.

Table 3. Parameters for DOEF.

6.3 Dynamic Clustering Algorithms Tested

We compare the performance of four dynamic clustering algorithms in this paper. These include two existing algorithms, dynamic statistical tunable clustering (DSTC), detection & re-clustering of objects (DRO) and two new OPCF algorithms, dynamic greedy graph partitioning (OP-GP) and dynamic probability ranking principle (OP-PRP).

The parameters used for the dynamic clustering algorithms are shown in table 4. In order to tune the clustering algorithms we have tested a range of different parameter settings for each algorithm in each simulation. Then for each algorithm we have used the set of parameters that gives the best overall result for all the simulations. We found algorithm performance was not very sensitive to parameter settings across different simulations. This means the same set of parameters usually performed well for all simulations or none of the simulations. For a more detailed description of the DSTC and DRO parameters please refer to [3] and [5] respectively. Detailed descriptions of OPCF algorithm parameters can be found in section 5.

The dynamic clustering algorithms shown on the graphs in this paper are labeled as follows:

[8] All database objects can be used as traversal roots.

(a) DSTC parameters

Parameter	Description	Value
n	maximum entries in the observation matrix	200
n_p	number of observation periods after which a consolidated factor fc_{ij} becomes obsolete if the link (o_i, o_j) has not been detected.	1
p	the number of the current observation period is computed modulo p.	1000
T_{fa}	threshold value under which the number of accesses to individual objects is too small to be considered in elementary linking factors computation.	1
T_{fe}	threshold value under which elementary linking factors are not consider for updating consolidation factors.	1
T_{fc}	threshold value under which a consolidated linking factors are not considered significant.	1
w	weighting coefficient introduced to minimise significance of elementary observations relative to consolidated observations.	0.3

(b) DRO parameters

Parameter	Description	Value
MinUR	minimum usage rate.	0.001
MinLT	minimum loading threshold.	2
PCRate	page clustering rate.	0.02
MaxD	maximum distance.	1
MaxDR	maximum dissimilarity rate.	0.2
MaxRR	maximum resemblance rate.	0.95
SUInd	statistics update indicator.	true

(c) OPCF parameters

Parameter	Description	OP-PRP Value	OP-GP Value
N	number of objects referenced between clustering analysis iterations.	200	200
CBT	clustering badness threshold.	0.1	0.1
NPA	number of pages analysed during each clustering analysis iteration.	50	50
NRI	number of re-organisation iterations after each analysis iteration.	10	10
ETT	external tension threshold.	–	0.2

Table 4. Parameters used for dynamic clustering algorithms.

- **NC**: No Clustering;
- **DSTC**: Dynamic Statistical Tunable Clustering;
- **OP-GP**: OPCF (greedy graph partitioning);
- **OP-PRP**: OPCF (probability ranking principle);
- **DRO**: Detection & Re-clustering of Objects.

All the results presented in this paper are in terms of total I/O. Total I/O is the sum of transaction read I/O, clustering read I/O and clustering write I/O. Thus, the results give an overall indication of the performance of each clustering algorithm, including each algorithm's clustering I/O overheads.

7 Simulation Results

In this section we report the results of simulations comparing the performance of two existing highly competitive dynamic clustering algorithms (DSTC, DRO) and two new algorithms produced using the OPCF framework (OP-PRP, OP-GP).

7.1 Varying Buffer Size

In this section we report the effects of varying buffer size on the performance of the dynamic clustering algorithms. The results are shown on figure 8. The OPCF algorithm OP-GP offers best overall performance. OP-GP performs particularly well when the buffer size is small. At a buffer size of 0.4MB, OP-GP produces 34% less I/O than DRO and 48% less I/O than DSTC. The performance advantage can be attributed to OP-GP's opportunistic behaviour. OP-GP reduces many clustering read and write I/Os by limiting re-clustering to only in-memory pages. There are two reasons why opportunism is particularly beneficial at small buffer sizes. First, at small buffer sizes a page read from disk for clustering purposes (non-opportunistic behaviour) is less likely to be referenced by the user application before it is evicted. Thus the cost of the clustering read I/O is less likely to be absorbed by user applications. Second, at small buffer sizes, read I/O for clustering purposes are more likely to cause premature evictions of cache resident pages.

OP-GP outperforms OP-PRP for all buffer sizes tested. This is because OP-PRP does not take inter-object relationships into consideration when clustering. However, it is encouraging to note that OP-PRP (a very simple algorithm, with very small statistics overheads) can outperform complex algorithms like DSTC and DRO for a variety of buffer sizes.

At large buffer sizes (above 1MB) OP-GP maintains its lead over DRO by producing between 5% and 15% less I/O. The performance advantage of OP-GP can be attributed to a combination of opportunism and the ETT threshold. The ETT threshold limits clustering write overheads by limiting the scope of clustering to pages that will benefit the most from re-clustering.

DSTC performs very poorly, worse than no clustering, for most buffer size settings. The poor performance can be attributed to DSTC's high clustering overheads. The high clustering overheads are caused by two factors. First, DSTC is not opportunistic and thus generates a lot of clustering read I/O. Second, DSTC's tuning parameters lack flexibility. In our simulations, increasing the parameters T_{fa}, T_{fe} and T_{fc} by the smallest possible increment causes the algorithms to go from over excessive clustering to almost no clustering (both settings result in about the same performance). The results shown are when the more aggressive clustering setting is used. We have tested all algorithms extensively with different parameters settings and only used the settings that gave the best overall performance for each algorithm.

7.2 Varying Hot Region Size

In this section we examine the effect of changing hot region size on the dynamic clustering algorithm performance. The hot region probability of access is set to 80% for the reasons explained in section 6.2. The results for two buffer size settings of 1MB and 4MB are reported in figures 9 (a) and (b), respectively. The results in this simulation show OP-GP providing best overall performance. We attribute this mainly to its use of opportunism and its use of sequence tension to model access dependencies between objects.

At the small buffer size of 1MB, OP-GP's performance degrades at a slower rate than DRO, the best existing dynamic clustering algorithm. The reason for this is that increasing the hot region size has the effect of increasing the working set size. A larger working set means more objects will have higher usage rates. This in turn means DRO, which only clusters pages with usage rates above the $MinUR$ threshold, clusters more aggressively as working set size increases. As the hot region size increases, a larger portion of the working set is disk resident. DRO, which is not opportunistic, performs more clustering read I/O since a larger portion of the working set is disk resident.

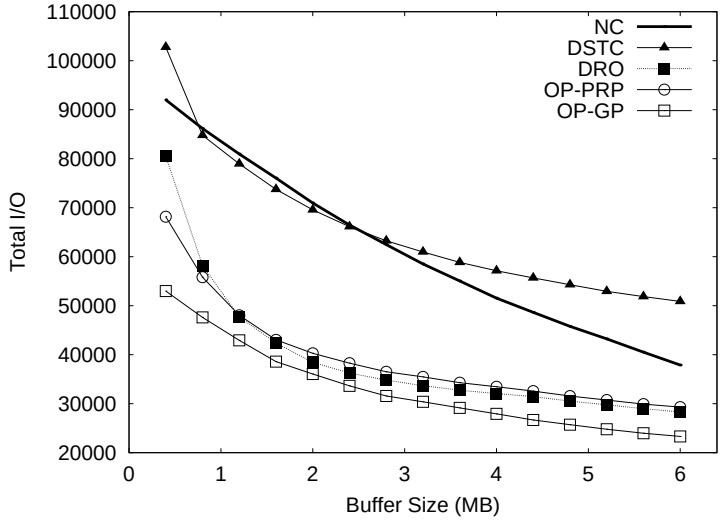


Fig. 8. Buffer size versus total I/O.

DRO also performs more clustering write I/Os as the hot region size increases. This is because as more pages are loaded into memory for clustering purposes, more dirty pages are evicted (working set does not fit in memory). Dirty page evictions causes write I/O. In contrast, OP-GP's opportunistic behaviour combined with its *bounded* scope of re-organisation results in a smaller clustering I/O footprint, for both small and large hot region sizes.

At the larger buffer size of 4MB (figure 9 (b)), almost all of the working set fits in memory (even when the hot region size is 9% of database size). In this environment, DRO's performance approaches that of OP-GP as the hot region size increases. This is because as more of the working set fits in memory, DRO's lack of opportunism is less damaging to its performance. Since most of the pages it wants to cluster are memory resident, less clustering read I/O is required. Clustering write I/Os are also decreased due to fewer dirty page evictions.

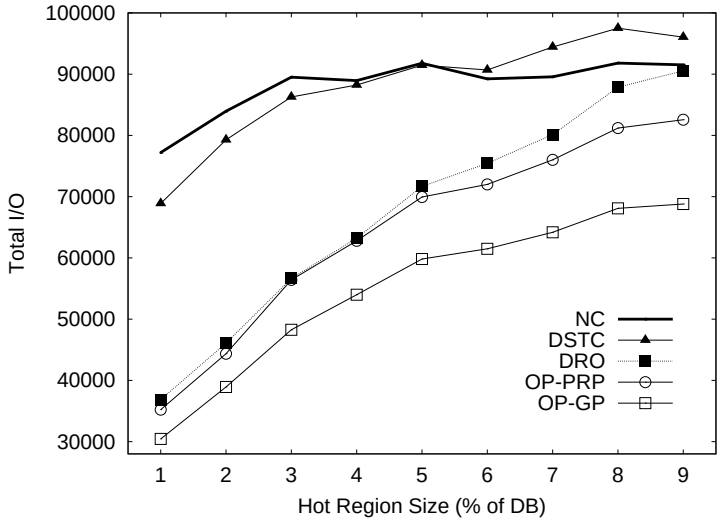
7.3 Varying Hot Region Access Probability Size

In this section we report the effects of changing hot region access probability. The hot region is set to 3% of the database size for reasons explained in section 6.2. The results of two buffer size settings of 1MB and 4MB are reported on figure 10 (a) and (b), respectively.

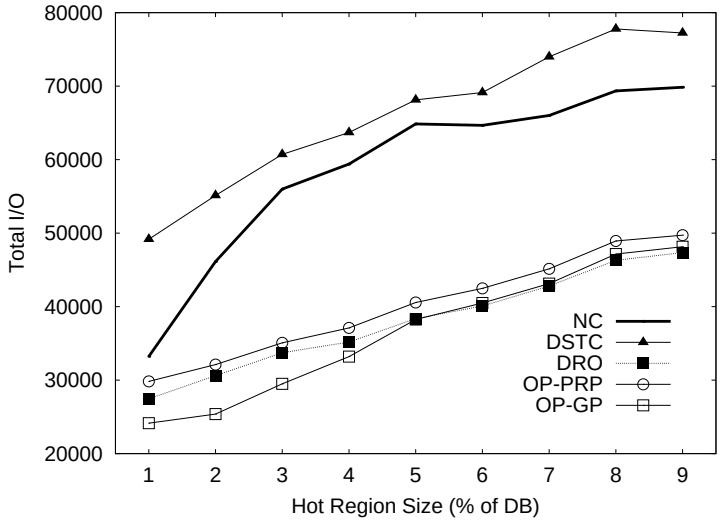
The results for 1MB buffer size show OP-GP offering the best performance at small hot region access probabilities. However, when hot region access probability is high, OP-GP's performance degrades to be worse than DRO. The reason for OP-GP's success at low access probabilities is OP-GP's ETT threshold and opportunism protects it from aggressive re-clustering of the cold region. In contrast, DRO does not have these features. Thus at these settings DRO finds it difficult to distinguish between the hot and cold region (the difference between access probability of hot and cold region is small), and consequently re-clusters much of the cold region aggressively. The result is that DRO generates a lot of clustering read and write I/O overheads for marginal transaction read I/O gains. However, when hot region access probability is high, OP-GP's ETT threshold starts to work against it. This is because ETT restricts re-clustering (even in the hot region) to those pages that give the largest performance improvement. In contrast, DRO, which aggressively re-clusters the hot region, benefits from improved transaction read I/O gains. At high hot region access probabilities, DRO no longer has difficulty separating the hot and cold region for re-clustering. Thus it no longer spends clustering I/O resources re-clustering the cold region.

The results for the 4MB buffer size (see figure 10 (b)) again show OP-GP offering best performance when hot region access probability is low. When hot region access probability is high, OP-GP and DRO perform approximately the same. The reason for OP-GP's superior performance at low hot region access probabilities is the same as for the 1MB buffer size results. However, at high hot region access probabilities, the larger buffer size is more forgiving for OP-GP's lack of aggression in re-clustering the hot region. Thus OP-GP's performance no longer degrades to worse than DRO at high hot region access probabilities.

DSTC, which has been set to re-cluster aggressively (at low aggression settings it performs almost no re-clustering, and there is no middle ground), shows rapid performance improvement when hot region access probability increases. This is because at high hot region access probabilities, aggressive re-clustering is confined to primarily the hot region



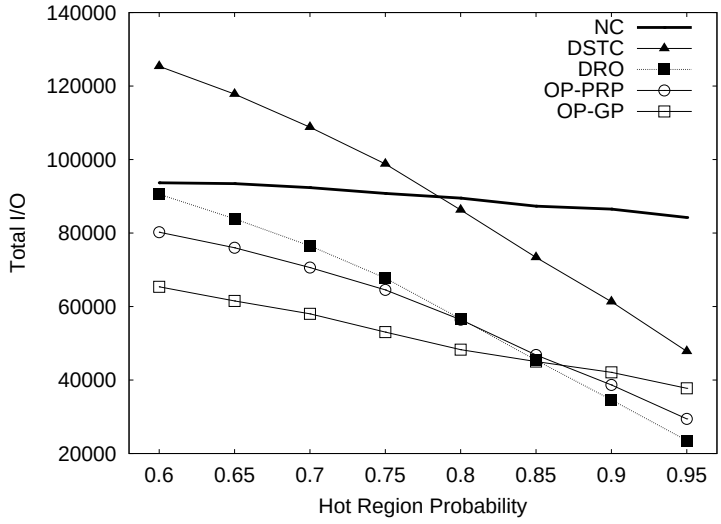
(a) 1MB buffer size.



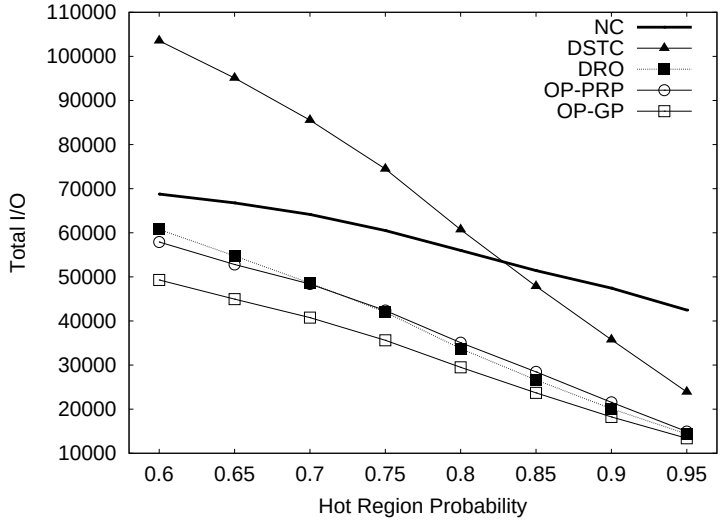
(b) 4MB buffer size.

Fig. 9. Hot region size versus total I/O.

(the cold region rarely gets touched and thus is rarely re-clustered). This fact means DSTC's clustering I/O overheads rapidly diminish as the hot region access probability increases.



(a) 1MB buffer size.



(b) 4MB buffer size.

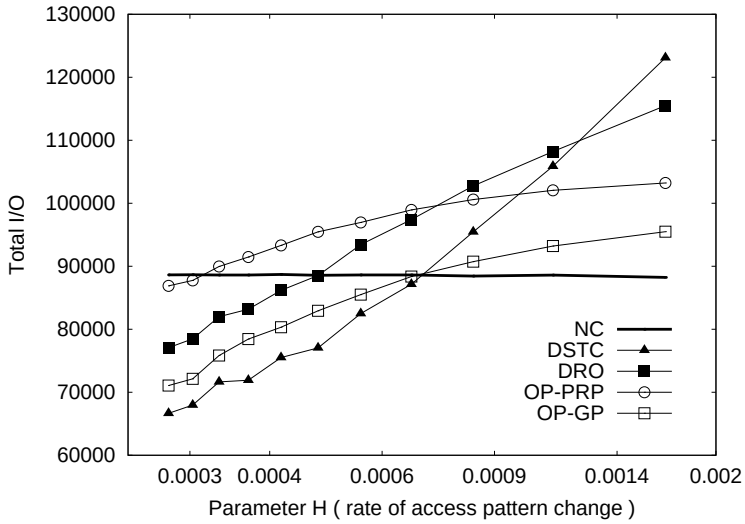
Fig. 10. Hot region probability versus total I/O.

7.4 Moving Window of Change

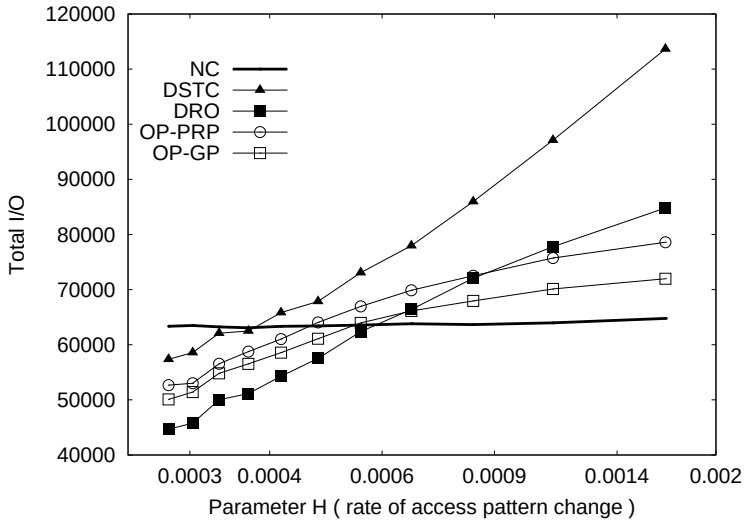
In this simulation, we used the *moving window of change* protocol to test each of the dynamic clustering algorithms' ability to adapt to changes in access pattern. The DOEF settings used are shown in table 3. In this simulation we varied the parameter H, rate of access pattern change. We report the results using 1MB and 4MB buffer sizes. The results are shown on figure 11 (a) and (b), respectively. The x-axis in the graphs are in log_2 scale.

The general observation is that OP-GP offers poor performance when the rate of access pattern change is small but offers best performance (when compared to the other dynamic clustering algorithms) when the rate of access pattern change is high. The poor performance of OP-GP under small access pattern changes is that OP-GP has not been designed for this vigorous style of change (where the hot region changes suddenly from 0.8 probability of access to 0.0006 probability of access). The other algorithms, DSTC and DRO, periodically delete gathered statistics which

makes them cope much better in this style of change. However, when the rate of access pattern change is very high, DSTC and DRO suffer from over aggressive re-clustering. Both DSTC and DRO do not place hard bounds on the scope of re-organisation. This causes them to re-cluster vigorously at high rates of access pattern change (the clustering on many pages appear to be outdated). Much of the hard re-clustering work goes to waste, since re-clustered pages quickly get outdated. In contrast, OP-GP and OP-PRP, which place hard limits on the scope of re-organisation, perform better at higher rates of access pattern change.



(a) 1MB buffer size.



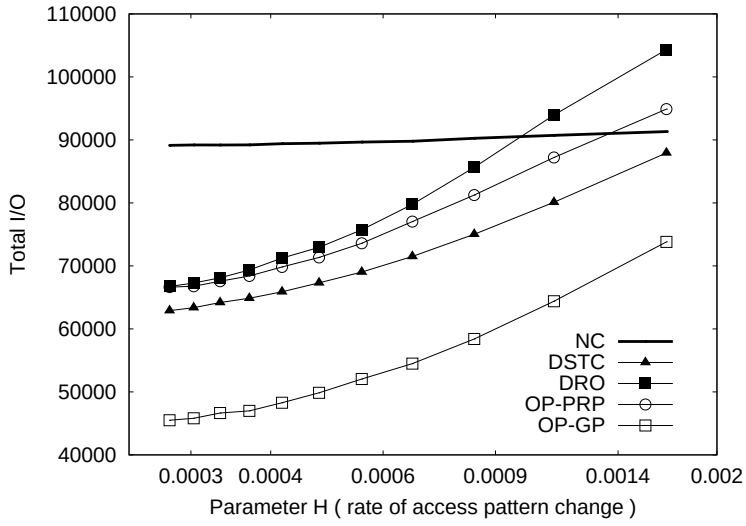
(b) 4MB buffer size.

Fig. 11. Moving window of change

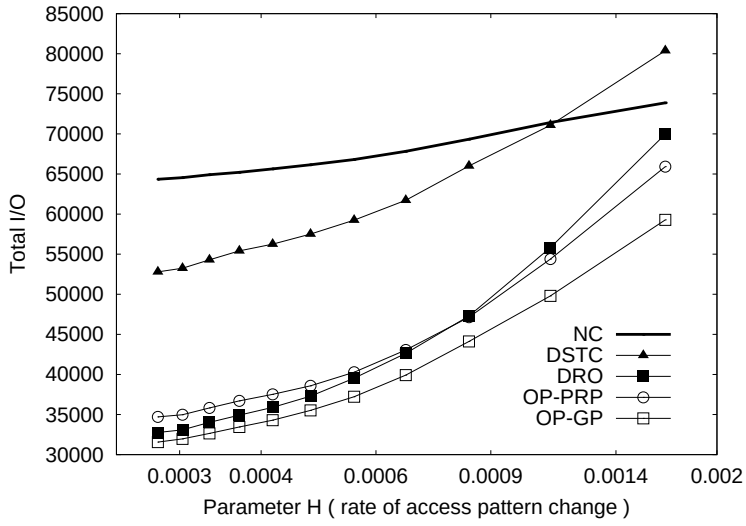
7.5 Gradual Moving Window of Change Simulation

In this simulation, we used a less vigorous style of access pattern change, the *gradual moving window of change* protocol. Unlike the previous simulation, the hot region cools gradually instead of suddenly. The DOEF settings used are again shown in table 3. In this simulation we varied the parameter H, rate of access pattern change. We report the results using 1MB and 4MB buffer sizes. The results are shown on figure 12 (a) and (b), respectively. The x-axis in the graphs are in log_2 scale.

The results show OP-GP consistently outperforming the other dynamic clustering algorithm when this gradual style of change is used. This is due to OP-GP's policy of not deleting old statistics, which is beneficial instead of detrimental to clustering quality in this simulation. The gradual cooling of the hot region means that as the hot region cools a lot of residual heat remains. OP-GP, which never deletes old statistics, continues to re-cluster the slowly cooling hot region.



(a) 1MB buffer size.



(b) 4MB buffer size.

Fig. 12. Gradual moving window of change.

8 Comparisons

In this section we summarize the differences between algorithms produced by OPCF and existing dynamic clustering algorithms.

The main difference between OPCF and DROD [24] is that DROD requires statistics of all objects in the object base. This means there is a lot of memory that needs to be allocated for storing statistics, which in turns means there is less memory for the data which can cause more disk I/O to occur. In contrast OPCF algorithms only require statistics of objects in memory.

The main difference between OPCF and the three existing algorithms StatClust [11], DRO [5] and DSTC [3] is that these existing algorithms do not bound the number of pages involved in reclustering in each re-organisation iteration. Bounding the number of pages involved in reclustering is very important since that limits the disruption to the database application. In contrast to the three existing dynamic clustering algorithms, OPCF only reclusters a bounded number of pages in each re-organisation iteration.

9 Conclusions

In this paper we have presented OPCF, a generic framework which when applied to static clustering algorithms, can produce dynamic clustering algorithms that possesses the two desirable properties of *opportunism* and *prioritisation of clustering*. In addition, application of the framework is straightforward and yet it produces clustering algorithms that outperform an existing highly competitive dynamic clustering algorithm, DSTC [3], in a variety of situations.

The results reported in this paper test the dynamic clustering algorithms in a wide variety of situations, including various buffer sizes, hot region sizes, hot region access probabilities and various rates of access pattern change. The results show that the OPCF algorithm, OP-GP, offers best performance in most situations. OP-GP derives its performance advantage mainly from its opportunistic behaviour and its use of sequence tension to model access dependency between objects. Opportunism reduces OP-GP's clustering I/O overhead, while sequence tension allows OP-GP to achieve high clustering quality. The combination of low clustering I/O overhead and high clustering quality gives OP-GP the best overall performance.

OP-GP is the most consistent performer when access pattern change is introduced. OP-GP clearly outperforms all other algorithms when the more moderate *gradual moving window of change* protocol is used. For the more vigorous *moving window of change*, OP-GP outperforms all other algorithms when the rate of access pattern change is high. However, when access pattern change is low, it loses out to DSTC at the small buffer size of 1MB and loses out to DRO at the larger buffer size of 4MB. This can be attributed to OPCF algorithms' policy of never deleting old statistics. Introducing a mechanism for deletion of old statistics seems a simple way to fix the problem. However, the deletion must be done carefully, since keeping old statistics is desirable when the more moderate gradual moving window of change protocol is used. Alternatively, a statistics aging policy may be introduced. A possible policy may be to give more recent statistics higher weights. We leave finding a suitable deletion policy and statistics aging policy to future work.

In other future work, we would like to explore the performance of transforming other static clustering algorithms using OPCF and see how they perform in relation to the algorithms presented in this paper.

Acknowledgements

The authors wish to acknowledge that this work was carried out within the Cooperative Research Center for Advanced Computational Systems established under the Australian Government's Cooperative Research Centers Program.

We would like to thank Jerome Darmont for creating VOODB and making the sources publicly available. We would also like to thank Jerome for creating OCB, the benchmark used to conduct the simulations presented in this paper. Finally we would like to thank Luke Kirby and Eliot Moss 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. BULLAT, F., AND SCHNEIDER, M. Dynamic clustering in object databases exploiting effective use of relationships between objects. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1996), Linz, Austria* (1996), Springer, pp. 344–365.
4. 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.
5. DARMONT, J., FROMANTIN, C., REGNIER, S., GRUENWALD, L., AND SCHNEIDER, M. Dynamic clustering in object-oriented databases: An advocacy for simplicity. In *Proceedings of the International Symposium on Object and Databases* (June 2000), vol. 1944 of *LNCS*, pp. 71–85.

6. 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.
7. 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.
8. DEUX, O. The O_2 system. *Communications of ACM 34*, 10 (1991), 34–48.
9. 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.
10. 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.
11. GAY, J.-Y., AND GRUENWALD, L. A clustering technique for object oriented databases. In *Proceedings of the International Conference on Database and Expert Systems (DEXA 1997)* (September 1997), vol. 1308 of *Lecture Notes in Computer Science*, Springer, pp. 81–90.
12. GERLHOF, A., KEMPER, A., AND MOERKOTTE, G. On the cost of monitoring and reorganization of object bases for clustering. In *ACM SIGMOD Record* (1996), vol. 25, pp. 28–33.
13. 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.
14. 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.
15. HE, Z., AND DARMONT, J. Dynamic object evalutaion framework (DOEF). In *Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA 2003)* (2003), pp. 662–671.
16. HUDSON, E., AND KING, R. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. In *ACM Transactions on Database Systems* (September 1989), pp. 291–321.
17. KERNIGHAN, B., AND LIN, S. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* (1970), 291–307.
18. KNAFLA, N. *Prefetching Techniques for Client/Server, Object-Oriented Database Systems*. PhD thesis, University of Edinburgh, 1999.
19. MCIVER, W. J. J., AND KING, R. Self-adaptive, on-line reclustering of complex object data. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1994), Minneapolis, Minnesota* (1994), R. T. Snodgrass and M. Winslett, Eds., ACM Press, pp. 407–418.
20. SINGHAL, V., KAKKAD, S. V., AND WILSON, P. R. Texas: An efficient, portable persistent store. In *Proceedings of the International Workshop on Persistent Object Systems (POS 1992)* (1992), pp. 11–33.
21. TSANGARIS, E.-M. M. *Principles of Static Clustering For Object Oriented Databases*. PhD thesis, University of Wisconsin-Madison, 1992.
22. TSANGARIS, M. M., AND NAUGHTON, J. F. A stochastic approach for clustering in object bases. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1991)* (1991), pp. 12–21.
23. TSANGARIS, M. M., AND NAUGHTON, J. F. On the performance of object clustering techniques. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1992)* (1992), pp. 144–153.
24. WIETRZYK, V. S., AND ORGUN, M. A. Dynamic reorganisation of object databases. In *Proceedings of the International Database Engineering and Applications Symposium* (August 1999), IEEE Computer Society.
25. WILSON, P. R., JOHNSTONE, M. S., NEELY, M., AND BOLES, D. Dynamic storage allocation: A survey and critical review. In *International Workshop on Memory Management* (Kinross, Scotland, UK, Sept 1995).