An Alternative approach to Temporary Memory Management in Databases using Object Oriented Systems

Sunil Kr Pandey, Dr. G.P.Singh**, Dr. Vineet Kansal**
*, ***Institute of Technology & Science, Mohan Nagar, Ghaziabad

**Govt. Dungar College, Bikaner, Rajasthan

Abstract
Regardless of the supremacy of relational database management systems (RDBMS) in the databases, object-oriented database management systems (OODBMS) continue to play crucial role in the management of data. Generally, the complex data are often found in telecommunications, business, engineering and web based applications. The most common approach of accessing such data is navigation. However, the approach of navigational access of data has the potential of generating excessive disk IO because objects in the path of navigation may be placed in different disk pages. Excessive disk IO is becoming increasingly undesirable because disk IO performance improves at only 5-8% per year whereas CPU performance doubles approximately every 18 months. Thus disk IO is likely to be a bottleneck in an increasing number of OODB applications. This paper focuses on reducing disk IO effects to improve OODBMS performance. In database environment effective buffer management of the main memory is the key in increasing efficiency through reducing the disk IO bottleneck in OODBMSs. There has been much existing work, namely in the areas of: *static clustering*; *dynamic clustering*; *buffer replacement*; and *pre-fetching*. All of these techniques can be used together in a complimentary manner. Most existing research has focused on finding the best solution for each area with little regard on how solutions from the different areas affect each other. We believe synergy exists between the areas, and that exploiting the synergy leads to the best overall solution. This paper focuses on exploring whether synergistic techniques are both feasible to implement and outperform their non-synergistic counterparts.

Key words:
Temporary Memory Management, Databases, Object Oriented System

1 Introduction

Continuous growth in the size of the databases and requirement of accessing data residing at different places in different tables, one of the most effective techniques for performance enhancement is considered to be clustering [Gerlhof et al. 1996]. The reason behind this is the navigational object accesses in an object oriented database. Consequently, related objects are often accessed consecutively. By grouping related objects onto the same disk page in an object oriented database environment disk IOs can be reduced. In addition to reduced IO, clustering also uses cache space more efficiently by reducing the number of unused objects that occupy the cache. Periodical re-clustering allows the physical organization of objects on disk to more closely reflect the prevailing pattern of object access. The majority of existing clustering algorithms are static [Tsangaris 1992; Banerjeeet al. 1988; Gerlhof et al. 1993; Drew et al. 1990]. Static clustering algorithms require that re-clustering take place when the database is not in operation, thus prohibiting seamless and uninterrupted accessibility of databases. In contrast, dynamic clustering algorithms re-cluster the database while database applications are in operation. Applications that require round the clock access of Object oriented database and involve frequent updates in data access patterns may benefit from the use of dynamic clustering.
Generally from the study it is revealed that, following three properties are generally missing from most existing dynamic clustering algorithms. These properties include:

- The re-use of existing work on static clustering algorithms;
- The use of opportunism to minimize the IO path for re-organization; and
- A prioritization of re-clustering so the worst clustered pages are re-clustered first.

In spite of the works by many of the prominent researchers that exists on static clustering [Tsangaris 1992; Banerjee et al. 1988; Gerlhof et al. 1993; Drew et al. 1990], there has been little transfer of ideas into the dynamic clustering literature. In this paper we try to explore this omission which transforms existing static clustering algorithms into dynamic algorithms. The goal of dynamic clustering is to generate the minimum number of disk IOs for a given set of database application access patterns. As pointed out by the cost models, the clustering process itself may generate IO, loading data pages for the sole purpose of object base re-organization. However, most researchers have chosen to ignore these sources of IO generation and instead concentrate on developing the dynamic clustering algorithm that minimizes the number of transaction read IOs generated by the client processes.

Manuscript received October 5, 2010
Manuscript revised October 20, 2010

Opportunism eliminates clustering read IO by choosing in-memory pages for re-clustering.

2 Related Works

The re-organization phase of dynamic clustering can incur considerable operating cost. Two of the key overheads are increased write contention and IO. To reduce write contention, most dynamic clustering algorithms are designed to be incremental and thus limit the scope of re-organization. However, *DROD* [Wietrzyk and Orgun 1999] is the only algorithm that we are aware of that limits the scopes of reorganization so that only in-memory objects are re-clustered. Wietrzyk and Orgun [1999] 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 affected by the modification or insertion. Once the new placement is determined, only the objects in memory are re-organized 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).

The incremental nature of dynamic clustering requires that only a small portion of the entire database be re-clustered after each iteration. However, the choice as to which portion to re-cluster is where many existing algorithms differ. McIver and King [1994] suggest targeting the portion that was accessed after the previous re-organization. However, this may involve a very large portion of the database if the re-clustering is not triggered frequently. Wietrzyk and Orgun [1999] re-cluster affected objects as soon as an object graph modification occurs. They use a threshold mechanism[2] 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* [Gay and Gruenwald 1997] and *DRO* [Darmont et al. 2000] 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-organization 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-organization 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 [Bullat and Schneider 1996]. Therefore, even if the improvement is very small, a re-clustering of those pages that can be improved is triggered. 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-organization iteration by breaking the re-organization workload into re-clustering units and only re-organization one unit in each iteration.

A large body of work exists on static clustering algorithms [Tsangaris 1992; Banerjee et al. 1988; Gerlhof et al. 1993; Drew et al. 1990]. However, only relatively few static algorithms have been transformed into dynamic algorithms. McIver and King [1994] combined the existing static clustering algorithms, Cactis [Hudson and King 1989] and DAG [Banerjee et al. 1988], 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 [Tsangaris 1992]. Wietrzyk and Orgun [1999] develop a new dynamic graph partitioning clustering algorithm. However, they do not compare their dynamic graph partitioning algorithm with any existing dynamic clustering algorithm.

3 Preliminaries

In this section we first provide a formal definition of the problem we are attempting to explore, then outline the problem constraints and possible assumptions.

3.1 Problem Definition

Using the integrated cost model we now formally define the problem. The threads that we have are:

- This client thread (TC)
- Other client threads (OC)
- Dynamic clustering thread (DC)

Given a trace *ti*, an initial object to page mapping (initial clustering), a buffer replacement algorithm and an interleaving *xi(Tn)* , we seek the *dynamic clustering*

algorithm that minimizes the execution time *ET(xi (Tn) ti) of* of *ti* under *xi (Tn).* This is formulated as:

$$\text{Min}(ET(x_i(T_n),t_i)) = \text{Min}(\sum_{r=0}^{h} IO_{TCR}(r) + \sum_{r=0}^{h}(IO_{DCR}(r) + CPU_{DC}(r)) + \sum_{r=0}^{h} IO_{OT}(r))$$

Eq. - 1

Dynamic clustering has negligible effect on the amount of CPU time used by the client threads. The dynamic clustering thread is able to change the object-to-page mapping and thus has the potential to reduce the number of future read and write IOs. In addition, the dynamic clustering algorithm may slow down the system by generating read and write IO and consuming CPU resources.

Equation 1 can be further decomposed into:

$$IO_{OT}(r) = IO_{OCR}(r) + IO_{BW}(r)$$

Eq. - 2

We do not aim to produce dynamic clustering algorithms that reduce *IOBW*(r) by re-organizing objects in such a way that dirty objects are likely to be placed into the same page. This is because our focus is on read IO. However, we do aim to produce dynamic clustering algorithms that impose a small write IO footprint.

3.2 Constraints

This section lists two constraints placed on the dynamic clustering thread. The constraints limit the duration and frequency of re-clustering.

3.2.1 Limited Duration

This constraint limits the duration of each re-clustering iteration. A re-clustering iteration is defined as a period of *continuous* re-clustering activity. This is done by limiting continuous re clustering time to be shorter than a user defined *Tc* threshold of time units as follows:

Let*CCR*(i,j) be a period of time with continuous clustering related requests, where i^{th} and j^{th} references delimit the start and end of a clustering iteration.

$\forall i, j$ such that $CCR(i, j)$ is valid:

$$\sum_{r=i}^{j}(IO_{DCR}(r) + CPU_{DC}(r)) < T_c$$

Eq. - 3

3.2.2 Limited Frequency

This constraint ensures a minimum time for a client thread to work before it is interrupted by limiting the frequency of re-clustering. This is done by ensuring that client threads are not interrupted for at least a user defined threshold of *Tt* time units as follows:

Let *CTR*(i, j) be the time between successive re-clustering iterations, the i^{th} and j^{th} references delimit the start and end of a session which is not interrupted by clustering.

$\forall i, j$ such that $CTR(i, j)$ is valid:

$$\sum_{r=i}^{j}(IO_{TCR}(r) + IO_{OT}(r) + CPU_{OT}(r)) > T_t$$

Eq. - 4

Constraints 3 and 4 combine to limit the frequency and duration of re-clustering iterations.

3.3 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[3] do not benefit from clustering, we choose to focus our study on objects smaller than a page in size.
3. The patterns of object access after and before each re-organising iteration bare some degree of similarity.

4 Opportunistic Prioritized Clustering Framework (OPCF)

In this section we outline in detail the possible framework to solve the above problems, the Opportunistic Prioritized Clustering Framework (OPCF) (Zhen He, 2004); OPCF transforms static clustering algorithms into dynamic algorithms and provides them with the attributes of *opportunism, incrementality,* and *prioritization.* We begin by describing how OPCF achieves these attributes. We then define the steps of the OPCF framework.

4.1 Opportunism

OPCF introduces the opportunism property to minimize read IO overheads caused by the dynamic clustering thread. Thus opportunism attempts to achieve the following minimization:

$$\text{Min}(\sum_{r=0}^{h} IO_{DCR}(r))$$

Eq. - 5

OPCF achieves opportunism by restricting clustering to *in-memory* pages only.

4.2 Incrementality

OPCF limits the disruption caused by the dynamic clustering thread by *incrementally* re-organizing the database. This property of OPCF allows the dynamic clustering algorithm to meet the constraints of equations 3 and 4. OPCF achieves constraint 3 by placing a fixed bound on the number of pages re-clustered in each re-clustering iteration. Constraint 4 is accomplished by allowing users to control the frequency with which re-clustering is triggered.

4.3 Prioritization

Incrementality specifies that re-organization should be partitioned and only one portion of the database should be re-organized in each iteration. Prioritization specifies that the worst clustered portion should be targeted for re-organization in each iteration. We now explain the aim of prioritization by using equation-1. Prioritization aims to achieve large reductions of *IOTCR(r)* and *IOOCR(r)*costs while incurring only small *IODCR(r)* and *CPUDC(r)* costs. OPCF performs Prioritization by ranking pages in terms of quality of clustering and then limiting re-organization to a user-specified set of the worst clustered pages.

4.4 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 [Bullat and Schneider 1996] remove *selected* objects that are determined to need re-clustering from existing pages and place them into *new* pages. In order to create OPCF algorithms, a series of steps must be applied.

Define Incremental Re-organization Algorithm: In this step, a strategy is developed by which the existing static clustering algorithm is adapted to work in an incremental way. That is, at each iteration of re-organization, the algorithm must be able to operate within a limited scope.
Define Clustering Badness Metric: OPCF Prioritizes re-clustering by re-clustering the worst clustered pages first. This means that there must be a way of defining the quality of clustering at a page grain. 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 hot4 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 iteration,5 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 reorganization iteration a page is removed from the top of the priority queue and used to determine the scope of re-organization for that re-organization iteration. A user-defined number (NRI) of reorganization iterations are performed at the end of each clustering analysis iteration.
Define Scope of Re-organization: To limit the work done in each re-organization iteration of the dynamic clustering algorithm, a limited number of pages must be chosen to form the scope of re-organization. The scope of re-organization should be chosen in such a way that re organization 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-organization 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-organization. *Define Cluster Placement Policy*: Because OPCF works at a page rather than cluster grain, the initial stages of each re-organization iteration target a limited number of pages and so will, in general, identify multiple clusters, some of which may be small.6 The existence of clusters which are smaller than a page size raises the important issue of how best to pack clusters into pages. A simple way in which cluster analysis can be triggered in OPCF is by triggering cluster analysis when a user-specified number of objects (N) has been accessed. This is similar to the technique used in DSTC [Bullat and Schneider 1996]. However,
any other triggering method may be used, including triggering via an asynchronous thread (eg. for load balancing reasons).

5 Algorithms Generated Using OPCF

In this section we present two dynamic clustering algorithms generated using OPCF. We first describe two existing metrics that can be used to measure the quality of clustering. We then describe the static clustering algorithms from which our dynamic clustering algorithms are derived. Lastly, we describe in detail how OPCF is used to transform the static clustering algorithms into dynamic algorithms.

5.1 Two Metrics Used to Measure Quality of Clustering

Tsangaris and Naughton [1991, 1992] proposed two metrics for measuring the quality of an object clustering. working set size and long term expansion factor. *Working set size (WSS(M))* [Tsangaris and Naughton 1991] is a metric for locality that is cache replacement policy independent. WSS(M) is evaluated by taking Mframe requests, eliminating duplicates and computing the cardinality of the resulting set. Therefore, the larger the cardinality the fewer the duplicates and hence the lower the locality. A clustering algorithm that achieves a lower value for this metric will perform well on workloads that traverse a small portion of the database starting with a cold cache.

Long term expansion factor EF [Tsangaris and Naughton 1992] is an indicator of the steady state performance of an object clustering algorithm when the cache size is large. *EF* is the ratio of pages accessed in the steady state (*N*) to the number of pages that would be required ideally to pack all active objects (*n*). 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.

5.2 Static Probability Ranking Principle

The static probability ranking principle (PRP) algorithm [Tsangaris 1992] is the simplest sequence-based clustering algorithm. Sequence-based clustering [Banerjee et al. 1988; Drew et al. 1990; Tsangaris 1992] 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 graph7 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.
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 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 [Tsangaris 1992].
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.

5.3 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.
Incremental Re-organization 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-organization 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 in 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 both hot objects and 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:

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

Eq. - 6

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-organization: The scope of each re-organization 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 following figure-1 for an example.
This definition of scope of re-organization 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. The time complexity of this algorithm per re-organization iteration is *O(ns log ns)*, where *ns* is the number of objects within the scope of reorganization. The time complexity is determined by the sorting of objects within the scope of re-organization.

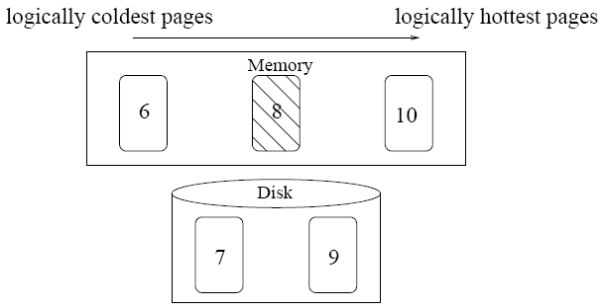

Figure 1: In this example the currently worst clustered page is 8, so the scope of reorganization for opportunistic dynamic PRP is pages 6, 8 and 10 (where page numbers reflect heat-order). If opportunism is not used, the scope would be pages 7, 8 and 9.

5.4 Static 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 *strucutural reference* traversal or the transition probabilities (*ˆP (x, y)*) of the SMC metric.
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) [Kernighan and Lin 1970], 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)[Gerlhof et al. 1993] 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 [Tsangaris 1992] 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.

5.5 Dynamic Graph Partitioning

This section outlines how we use OPCF to transform static graph partitioning algorithms into dynamic algorithms.

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

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

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. By contrast, the non-opportunistic algorithm also counts edge weights that crosses page boundaries onto disk pages.

Scope of Re-organization: The scope of re-organization 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-organization to the pages that will benefit the most from the re-organization. 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-organization. See figure 2 for an example:

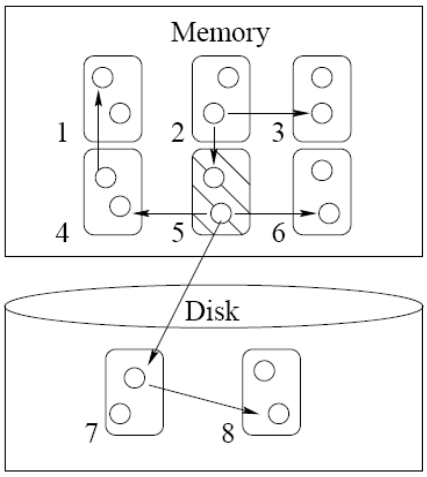

Figure 2: In this example the worst clustered page is 5 and the scope of re-organization for opportunistic dynamic graph partitioning are pages 2, 4, 5 and 6. The scope of re-organization for non-opportunistic dynamic graph partitioning are pages 2, 4, 5, 6 and 7.

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..
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(es log es + ps log ps)*, where *es* is the number of edges in the scope of re-organization and *ps* 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 *es* is much larger than *ps* and therefore in most cases the time complexity is *O(es log es)*.

6 Existing Dynamic Clustering Algorithms

In this section we describe in detail two existing dynamic clustering algorithms that are used in our performance study.

6.1 Dynamic Statistical and Tunable Clustering (DSTC)

DSTC is an existing dynamic clustering algorithm [Bullat and Schneider 1996] 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 minimize 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-organization: Using the information in the updated consolidated matrix, new clusters are discovered or existing ones are updated. In order to achieve incremental result the re-organization work is broken up into small fragments called clustering units.
Physical Clustering Organization: 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.

DSTC uses sequence tension information to model access dependencies between objects. DSTC is not an *opportunistic* clustering algorithm since its scope of re-organization can be objects that are currently residing on disk. DSTC exhibits a small degree of Prioritization 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.

6.2 Detection & Re-clustering of Objects (DRO)

Learning from the experiences of DSTC and StatClust [Gay and Gruenwald 1997], DRO [Darmont et al. 2000] is designed to produce less clustering IO 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 IO 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, in the discussed 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-organization iteration, whereas DRO has no such limit. DRO Prioritizes clustering by breaking up the database into pages that need re-clustering (according to thresholds) and pages that do not. This method of Prioritization 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-organization iteration).

Conclusion

The main conclusion of this paper is that simple synergistic frameworks can produce algorithms that provide significant performance gains when compared to existing non-synergistic algorithms. Furthermore the performance gains are across a wide variety of different situations with our guiding principles of *synergy*; *generality*; and *simplicity*. The preliminary results of this paper show that there is much potential in the synergistic approach to buffer management and suggests that perhaps the next big breakthrough in reducing the disk IO bottleneck in OODBMSs lies in synergistic buffer management techniques.

References:

[1] AILAMAKI, A., DEWITT, D. J., HILL, M. D., AND WOOD, D. A. 1999. 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. (p. 1)

[2] ARNOLD, A. O. 1978. Probability, Statistics, and Queuing Theory, with Computer Scient Application, Academic Press (pp. 21, 22)

[3] BANERJEE, J., KIM, W., KIM, S. J., AND GARZA, J. F. 1988. Clustering a DAG for CAD databases In IEEE Transactions on Software Engineering, Volume 14 (November 1988), pp. 1684.1699. (pp. 25, 26, 28, 33, 53, 54)

[4] BELADY, L. A. 1966. A study of replacement algorithms for a virtual-storage computer, IBM Systems Journal 5, 2 (pp. 54, 55, 63)

[5] BENZAKEN, V. AND DELOBEL, C. 1990. Enhaning performance in a persistent object store: Clustering strategies in o2. Technical Report 50-90 (August), Altair (p. 54)

[6] BERNSTEIN, P. A., PAL, S., AND SHUTT, D. 1999. Context-based perfecting for implementing objects on relations In Proceedings of the International Conference on Very Large Databases (VLDB 1999) (Sept 1999), pp. 327.338, Morgan Kaufmann (pp. 70, 71, 72)

[7] BULLAT, F. AND SCHNEIDER, M. 1996. Dynamic clustering in object databases exploiting effective use of relationships between objects, Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1996), Linz, Austria (1996), pp. 344.365. Springer. (pp. 3, 27, 28, 31, 32, 37, 44)

[8] CAO, P., FELTEN, E. W., KARLIN, A. R., AND LI, K. 1995. A study of integrated perfecting and caching strategies, Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1995) pp. 188.197, (pp. 3, 72)