Integrated Buffer Management for Object-Oriented Database Systems

Zhen He

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

January 2004

Typeset in Palatino by $\TeX$ and $\LaTeX\,2_{\varepsilon}$.

Except where otherwise indicated, this thesis is my own original work.

Zhen He
10 January 2004

To my loving wife Ying and my supportive parents.

Acknowledgments

A PhD student is very lucky to have one inspirational supervisor. I had the privilege of two, Steve and Alonso. Working with these two inspirational supervisors have transformed me into a much more creative and critical thinker. For this invaluable gift I am deeply grateful. Steve spent huge amounts of time teaching me how to do research. His generosity with time started right from the beginning of my honours degree and extended right to the very end of my PhD. He is not only a great supervisor but also a great friend. I would like to thank Steve for everything he taught me and showed me both in my research life and outside. Alonso has an amazing ability to think up all sorts of strange ideas out of the blue. These ideas became the fuel for many of the important concepts of this thesis. I would like to thank Alonso for always making it to our weekly meetings even when he is bogged down with enormous amounts of work in his consultancy job. Both supervisors are extremely nice people who have treated me with great respect. I am treated more like a friend than a student.

I would like to give a special thanks to the other members of my supervisory panel, Peter and Ramesh. Peter is an excellent chair of my supervisory panel. He is always very organised, making sure everything is done on time and discussing research plans with me. A big thank you to Peter for his careful proof reading of my work at various stages of my PhD. I would like to thank Ramesh for the interesting discussions we had about this thesis. I would also like to thank him for proof reading the final draft of this thesis.

Next I would like to thank my parents. My parents have been extremely supportive of my PhD in every respect, from cooking right to algorithm development. They have given me encouragement 24 hours a day and 7 days a week. Their belief in me has given me the confidence to not only pursue but also realise my dreams. So to my parents I say thank you for your continuous love.

Lastly, but most importantly I would like to thank my wife Ying. Her support and care really taught me the meaning of love. In addition, her playful personality made my PhD a much more enjoyable experience.

Abstract

Despite the dominance of relational database management systems (RDBMS) in the database market, object-oriented database management systems (OODBMS) continue to play an important role in complex data management. Complex data are often found in telecommunications, business, engineering and web based applications. The most common style of accessing complex data is navigation. Navigational access can generate excessive disk IO because objects in the path of navigation maybe 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 thesis focuses on reducing disk IO effects to improve OODBMS performance.

Effective main memory buffer management is the key to 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 *prefetching*. 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 thesis focus on demonstrating synergistic techniques are both feasible to implement and outperform their non-synergistic counterparts.

We made general modifications to existing techniques to demonstrate the superiority of synergistic buffer management. There were three guiding principles behind our modifications: *synergy*; *generality*; and *simplicity*. *Synergy* refers to modifications that exploit synergies between the different buffer management areas. *Generality* refers to modifications that can be applied to a large range of existing algorithms. *Simplicity* refers to making the modifications easy and straightforward to apply. Following these guiding principles we developed three new frameworks: opportunistic prioritised clustering framework (OPCF); cache conscious clustering framework (C3); and path and cache conscious prefetching framework (PCCP). Each framework addresses the synergy between two different buffer management areas. Using the frameworks we developed a total of seven new buffer management algorithms. These algorithms were found to outperform existing algorithms in a variety of situations.

This thesis takes a first exploratory look into how OODBMS buffer management techniques can be enhanced by synergistic modifications. The preliminary results show that there is much potential in this approach and suggests that perhaps the next big breakthrough in improving OODBMS main memory buffer performance lies in such techniques.

Contents

Chapter 1

Introduction

The ever increasing demand for fast complex data storage and retrieval makes a strong case for OODBMSs' survival as an important database management technology. Complex data can be found in many places: databases used in the telecommunications industry; business or finanical database applications; engineering database applications; and databases used to store web data. OODBMSs are particularly suited to the management of complex data since they provide fast navigational access, efficient storage of class methods and efficient and natural storage of many-to-many relationships.

Complex data is most often accessed via navigation. Relational database management systems (RDBMSs) are poorly suited for fast navigational access since simple object graph navigations can often turn into joins of multiple tables when converted to queries on the relational schema. Object-relational database management systems (ORDBMS) offer better navigational performance by storing references between objects inside the relational tables. Object navigations can then proceed by de-referencing these references instead of executing multiple joins. However this approach still does not perform as well as OODBMSs which often store objects traversed together on the same disk page, thus generating less IO. In contrast, ORDBMSs typically do not store objects traversed together on the same disk page, they are often stored as tuples on different relational tables instead. Typically, tuples of the same table are stored together on disk. Thus one of the most attractive characteristics of OODBMSs is fast navigational access.

The ability for OODBMS to provide fast navigational access is conditioned on efficient main memory caching, which is made more important by the fact that disk IO performance improves at only 5-8% per year whereas CPU performance doubles approximately every 18 months. A consequence is that disk IO is likely to be a bottleneck in an increasing number of OODB applications. Thus the focus of this thesis is on reducing the effects of disk IO on the performance of OODBMSs.

It should also be noted that much recent research on performance optimisation of RDBMSs has been focused on the main memory bottleneck instead of the disk IO bottleneck [Ailamaki et al. 1999; Chen et al. 2001; Rao and Ross 1999; Rao and Ross 2000]. This is due to main memory becoming cheaper and sophisticated techniques for hiding disk IO latency in RDBMSs. Some database users now choose to set up their system so that the entire database fits in memory. This decision is typically based on cost/performance trade-offs. However, disk continues to remain cheaper than memory, and so in any cost/performance analysis, scalability will ultimately dictate the use of disk. In addition, in many database applications a very high percentage of data accesses are directed at a very small portion of the database[Gray and Putzolu 1987]. In such cases, it is more cost-effective to only store a small portion of the database in memory (the portion which has a very high percentage of data access). When the database is larger than memory, techniques for hiding disk IO are needed to ensure the system is not bottlenecked at the disk. Existing techniques for hiding disk IO in OODBMSs do not perform as well as their RDBMS counterparts. This is because navigational data accesses (often used in OODBMSs) are much harder to predict than index and table accesses in RDBMSs. The disk IO bottleneck in OODBMSs is thus a pressing research problem.

1.1 Buffer Management Techniques

There are four proven techniques of improving the IO performance of OODBMSs: *static clustering*; *dynamic clustering*; *prefetching* and *buffer replacement*. *Clustering* is the arrangement of objects into pages so that objects accessed close to each other temporally are placed into the same page. When a requested object is loaded from disk, other objects (on the same page) which are likely to be needed in the near future are also loaded. This in turn reduces the total IO generated. Clustering algorithms can be separated into two types, *static* and *dynamic*. In *static clustering*, re-organisation takes place when the database is offline. In contrast, *dynamic clustering* re-organises the database while it is online. *Prefetching* involves predicting the user's next disk page request and then loading that page into memory in the background. In this manner disk IO can be overlapped with CPU, thus reducing disk IO stall time. *Buffer replacement* involves the selection of a page to be evicted when the buffer is full. The page evicted ideally should be the page needed furthest in to the future. Selection of the correct page for eviction results in a reduction in the total IO generated by the system.

1.2 Our Approach

This thesis is concerned with the integration of the four buffer management techniques mentioned in the previous section, namely: static clustering; dynamic clustering; buffer replacement; and prefetching. Previously these four buffer management techniques have mainly been considered in isolation. In contrast this thesis establishes that *synergy exists between these four techniques and that exploitation of this synergy leads to improved performance.*

To demonstrate the superiority of synergistic buffer management techniques, we made general modifications to existing techniques. There were three guiding principles behind our modifications: *synergy*; *generality*; and *simplicity*. *Synergy* refers to developing modifications that exploit synergies between the different buffer management areas. *Generality* refers to developing general modifications which can be applied to a large range of existing algorithms. *Simplicity* refers to making the modifications easy and straightforward to apply. Following these guiding principles we developed three synergistic frameworks: opportunistic prioritised clustering framework (OPCF); cache conscious clustering framework (C3); path and cache conscious prefetching framework (PCCP). Each framework addresses the synergies between two different buffer management areas.

1.3 Existing Work on Integrated Buffer Management

Gerlhof and Kemper [1994] show the importance of *clustering quality* to the performance of *prefetching algorithms* in OODBMSs. They found for applications with high locality, when running on a well clustered database, prefetching often achieves only negligible gains. However they did not vary the buffer size or report the buffer size they used. This leads us to suspect that at high locality and good clustering the working set size of the benchmark used was smaller than the buffer size. In these conditions the database does not need to load from the disk, hence the negligible impact of prefetching. In contrast, our experimental results report varying buffer sizes and show cases in which the working set is both bigger and smaller than the buffer size. Additionally Gerholf and Kemper do not propose any algorithms that exploit the synergy between prefetching and clustering, they only report the affects that clustering has on prefetching.

Cao, Felten, Karlin, and Li [1995] study the implications of integrating *prefetching* and *buffer replacement* when perfect knowledge of the future access sequence is known. They argued that prefetching too early may be harmful since early prefetching results in early buffer replacement if the buffer is full. Early buffer replacement can be harmful since new and better replacement opportunities may open up as the program proceeds. Using this observation they develop two new integrated prefetching and buffer replacement algorithms called *aggressive* and *conservative*. These strategies were found to reduce application running time by up to 50% compared to no prefetching. However, their algorithms assumes *perfect knowledge* of future access sequence, an unrealistic assumption in the real world. Their study was done within the context of operating systems buffer management.

Bullat and Schneider [1996] proposed an integrated *dynamic clustering*, *buffer replacement* and *prefetching* algorithm. Their integrated policy used the concept of a 'cluster unit'. A cluster unit is a set of clustered objects that span one or more pages. They perform prefetching and buffer replacement at the cluster unit grain instead of the page grain. Alternatively prefetching and buffer replacement can occur at the 'moving cluster-window' grain — a sub-set of x pages of the cluster unit which has the requested object at its center. An important drawback of this approach is that often dynamic clustering algorithms produce cluster units that are *smaller* than a page in size. Also, most dynamic clustering algorithms do not arrange objects in cluster units, hence precluding the use of this approach. The cluster unit concept does not incorporate direction of navigation. Direction of navigation is very important for both prefetching and buffer replacement. In prefetching we are only interested in prefetching objects in the forward direction of navigation, similarly, in buffer replacement we are only interested in retaining in memory objects in the forward direction of navigation. Lastly, their paper only reported the results of a performance study comparing no clustering against their new dynamic statistical clustering technique (DSTC). The performance gains from integrating prefetching and buffer replacement were not reported.

1.4 Thesis Contributions

The primary contribution of this thesis is demonstrating that current buffer management techniques for OODBMSs can be made more effective by simple synergistic modifications. The evidence is presented via three empirical studies in which various existing buffer management techniques undergo synergistic transformations via simple and general transformation frameworks. In addition, an integrated buffer management cost model is developed.

A more detailed breakdown of the contributions of this thesis are:

- An integrated cost model that encapsulates the interaction between static clustering, dynamic clustering, buffer replacement and prefetching. This cost model serves two functions: it identifies the various ways the buffer management techniques affect system performance; and provides a starting point by which other researchers can approach the problem of developing synergistic techniques.

- The opportunistic prioritised clustering framework (OPCF), which transforms static clustering algorithms into dynamic clustering algorithms. The dynamic clustering algorithms produced by OPCF were given the *opportunism* and *prioritisation* properties. *Opportunism* refers to restricting re-organisation to in-memory pages only. *Prioritisation* refers to re-organising the worst clustered pages first. Cost models were used to explain the performance advantages of this approach. Two new dynamic clustering algorithms were produced using OPCF: dynamic probability ranking principle; and dynamic greedy graph partitioning. Experimental results show that dynamic clustering algorithms produced by OPCF outperform existing state-of-the-art dynamic clustering algorithms in a variety of situations (results shown in chapter 4).

- The cache conscious clustering framework (C3), which produces clustering algorithms that exploit the synergies between clustering and buffer replacement. A new cache conscious clustering algorithm called C3-GP was produced using the C3 framework. Experimental results show that C3-GP outperforms existing highly competitive static clustering algorithm in a variety of situations (results shown in chapter 5).

- The path and cache conscious prefetching framework (PCCP), which produces prefetching algorithms that exploit the synergy between prefetching and buffer replacement. In addition, PCCP introduces a new and novel method of incorporating path information to make prefetching both more accurate and more profitable via increased CPU and IO overlap. A new metric called the prefetch quality metric explains the intuition behind PCCP's superior performance. Four new prefetching algorithms are developed using PCCP. The results show that PCCP algorithms outperform highly competitive existing prefetching algorithms in both IO stall time and statistics storage overhead (results shown in chapter 6).

There are a number of areas this thesis does not cover. We provide a list of these as a way of defining the scope of this thesis and to provide starting points for interested researchers to extend the work reported in this thesis:

- The three frameworks OPCF, C3 and PCCP each address synergy for different pairs of the four buffer management areas (static clustering, dynamic clustering, buffer replacement and prefetching). However, techniques that incorporate synergies of all four areas are beyond the scope of this thesis. Interested researchers are encouraged to explore ways of integrating the three different frameworks in order to produce a single fully integrated framework. This fully integrated framework might then be used to produce integrated techniques that incorporate synergies between all four buffer management areas (see section 7.2 for a more detailed discussion).

- This exploration of buffer management synergy was not designed to be exhaustive. There may be numerous synergies between any pair of the buffer management areas. For example, there are numerous synergies between pre-fetching and buffer replacement: only non-memory resident pages are candidates for prefetching and buffer replacement algorithms play a large part in deciding which pages are more likely to be non-memory resident; early prefetching of pages may force the buffer replacement algorithm to evict in-memory pages earlier; buffer replacement algorithms may want to evict a wrongly prefetched page; etc. This thesis does not attempt to exhaustively document all possible synergies between buffer management areas. Instead it establishes that synergy does exist between these areas and that simple synergistic techniques can produce better performance.

1.5 Thesis Organisation

The remainder of this thesis is organised in six chapters. Chapter 2 describes the OODBMS concepts important to this thesis, along with any relevant design alternatives. The particular design alternatives chosen for this thesis are also documented. Chapter 3 presents the theoretical foundations (system model, cost models and practical reference models) used throughout this thesis to explain the intuition behind the performance advantages of the various synergistic buffer management techniques. Chapter 4 presents the work relating to the OPCF clustering framework. The work relating to the C3 clustering framework is presented in chapter 5. Chapter 6 documents the work relating to the PCCP prefetching framework. Finally, chapter 7 concludes the thesis by summarising the findings and conclusions made by this thesis. In addition, directions for future research are outlined.

Chapter 4 contains material published in the proceedings of the *1st International Symposium on Object and Databases* [He et al. 2000] and proceedings of the *14th International Database and Expert Systems Applications Conference (DEXA 2003)* [He and Darmont 2003]. Parts of the work presented in chapter 5 was published in the proceedings of the *12th International Database and Expert Systems Applications Conference (DEXA 2001)* [He and Marquez 2001].

Chapter 2

OODBMS Concepts

This chapter serves two important roles. It provides a description of the OODBMS concepts important to this thesis and defines the scope of this thesis within this context. This is accomplished by first defining each concept along with any references to the existing literature. Then wherever design alternatives arise, the alternative chosen by this thesis is outlined along with the reasons for the decision.

2.1 Object Oriented Programming Languages

The driving force behind the inception of OODBMSs is the desire to provide database functionality to objects from object-oriented programming languages (OOPL), eg. C++, Java, Smalltalk, etc. The OOPL object abstraction extends the data structure concept by including the following characteristics:

- *Structure:* Objects model real world entities which can consist of atomic components, objects (embedded objects) and references to objects. Atomic components are flat attributes like reals, integers, etc. Object references are 'pointers' to other objects.

- *Behaviour:* Methods are program code that manipulate and export object state. This contrasts with conventional languages that allow arbitrary code to manipulate data structures.

- *Type:* Type prescribes the *structure* and the *behaviour* of an object through the specification of its components and methods.

- *Identity:* Objects have a unique identity independent of the object state. The identity serves as a mechanism to name and locate the object.

Objects can be viewed as nodes of a directed graph, called an 'object graph'. The edges of the graph represent object references and are labeled with the names of the target objects. An example of an object graph is depicted on figure 2.1.

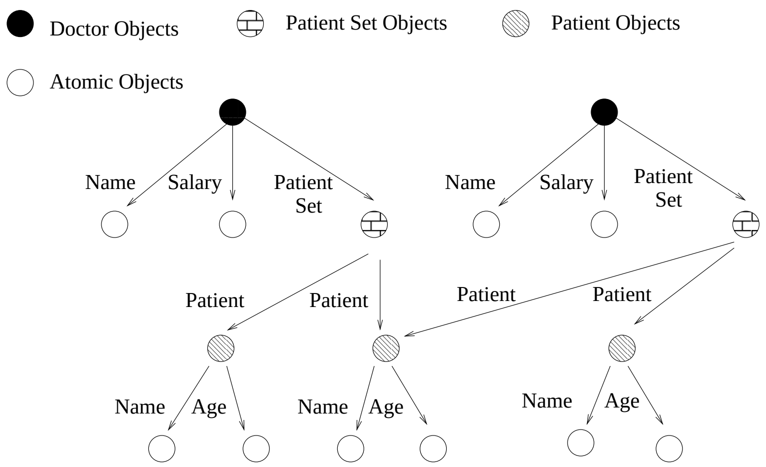


Figure 2.1: An example of an object graph. The nodes of the graph represent objects and the edges are labelled with the names of the target objects.

2.2 Object Identity

The way in which *object identity* is implemented plays a crucial role in determining the performance of an OODBMS. Broadly speaking there are three methods of implementing identity: *physical identifiers*; *logical identifiers*; and *structured identifiers*. *Physical identifiers* identify an object via the physical address of the object. This permits fast object access but makes object relocation costly. *Logical identifiers* identify an object via an unique identifier assigned to the object. The identifier is independent of the object location. The advantage of this approach is cheap object relocation. However the disadvantage of this approach is slower object access, since there is now an extra level of indirection. *Structured identifiers* contains both a physical and logical component. The physical component points to the region (such as a page) that the object resides in and the logical component identifies the object within the region. This approach allows objects to be accessed more cheaply than pure logical identifiers but more expensively than physical identifiers. Using this approach objects can be moved cheaply within a region but inter region object movements are more expensive.

Lakhamraju, Rastogi, Seshadri and Sudarshan [Lakhamraju et al. 2000] presents an efficient object relocation method for systems that use physical identifiers. They claim that their method "makes on-line re-organisation[1]feasible, with very little impact on the response times of concurrently executing transactions and on overall system throughput" [Lakhamraju et al. 2000].

Analysing the performance trade-offs of the different approaches is beyond the scope of this thesis. The work done in this thesis abstracts over this concern, that is, all algorithms proposed can be used in conjunction with either physical, logical, or

[1]Physical re-organisation of the database while the database is in operation.

structured identifiers. Our experimental studies also abstract over this concern by not simulating this system behaviour.

2.3 OODBMS Functionality

In addition to the OOPL features, OODBMS also provides the following functionality:

- *Persistence:* refers to the ability to maintain object state after the termination of program execution. OODBMSs allow large collections of objects to be stored into secondary "stable" storage, whereas OOPLs only support as many objects as will fit in main memory and swap space.
- *Storage Management:* refers to mechanisms used for the efficient storage of objects in main memory, disk and distribution across servers.
- *Concurrency Control & Recovery:* ensures that concurrent accesses to objects do not result in loss of data integrity. Object state is guaranteed to change in a consistent manner, and is immune to system failures.
- *Ad-hoc Query Facilities:* allow definition of declarative queries which perform operations on sets of objects.

In this thesis we focus on issues relating to storage management.

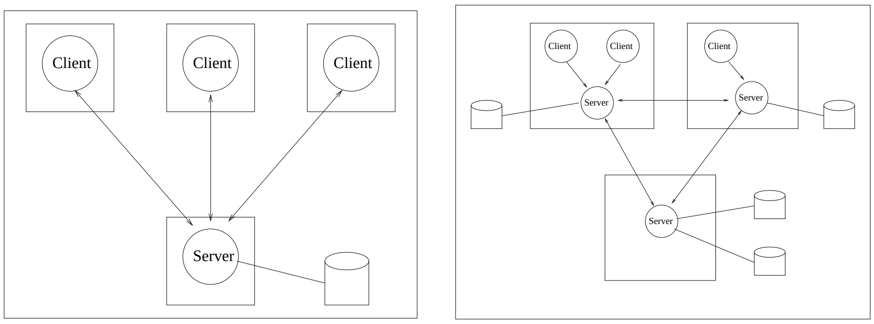
2.4 Architectural Issues

This section outlines various OODBMS architectural concepts and design alternatives. In addition, the particular design alternative chosen for this thesis is also explained and justified.

2.4.1 Client and Server Layers

OODBMS systems typically have the notion of a *client* and *server*. The *client* runs the user applications and the servers provide the database functionality. The client consists of the language run-time system, and the OODBMS run-time system necessary to communicate with the server. The *server* implements efficient stable storage for objects using the secondary storage, employing recovery, concurrency control, and other database protocols (eg. versioning, indexing, etc.).

Client programs mostly access objects sequentially (one at a time). One object at the time is 'visited' by de-referencing object pointers and thus 'navigating the object graph'. The exception occurs when ad-hoc queries are used, in which case operations are performed on sets of objects. When an object is accessed by the client, the OODBMS run-time provides the object (by issuing a request to the server) and verifies that the attempted operation on the object is allowed. Depending on the concurrency control protocols used, a read or write lock may be needed to complete the operation. When the object arrives from the server the suspended client computation resumes.



(a) Client/Sever (b) Peer-to-Peer

Figure 2.2: Alternative network models

2.4.2 Network Models

There are two standard network models for OODBMSs: *client/server*; and *peer-to-peer*:

- *Client/server:* In general this model is designed to run a client process on a small private workstation and communicate with one or a set of machines (with large disks) acting as a server. Figure 2.2 (a) shows a typical client/server network configuration. The advantage of this model is that client workstations do not need to perform much processing and thus can be relatively 'thin' machines. However, the disadvantage is the servers need to service requests from all clients and thus can become a bottleneck when the number of clients is large. Examples of client/server OODBMSs are Exodus [Carey et al. 1986], ObjectStore [Lamb et al. 1991], and O_2 [Deux 1991].

- *Peer-to-peer*: In this model every workstation on the network has a server process and any number of client processes running. The storage of the data is distributed across the machines. Figure 2.2 (b) shows a typical peer-to-peer network configuration. The advantage of this approach is that local data (data primarily used by a local client) can be stored on local servers and thus reduces access costs and decentralises data storage. Decentralised data storage means the server is less likely to be a bottleneck. The disadvantage of this approach is increased system complexity. Examples of peer-to-peer OODBMS are SHORE [Carey et al. 1994] and Platypus [He et al. 2000].

This thesis examines performance issues surrounding disk IO optimisation for a stand-alone single node of the peer-to-peer network model. This way we remove the network and remote caching behaviour of the general peer-to-peer network model and can thus focus our attention on reducing the effects of disk IO. However the techniques developed in this thesis, although designed for this more restricted model, can be generalised to work with the standard peer-to-peer and client/server network models (see section 7.2). The client/server network model is a special case of the peer-to-peer network model (since each node of the peer-to-peer network model can have zero or more clients and zero or more servers). In this thesis the stand-alone peer-to-peer configuration considered has one server and multiple clients all sharing a common cache (see section 3.1 for a more detailed description).

2.4.3 Granularity of Caching

In an OODBMS, main memory caches are used extensively to reduce network and IO costs. It is for this reason that we have decided to focus on main memory caches, however it is important to note that as the performance discrepancy between main memory and upper level caches widens, more work needs to be done at the higher level of caching. This important topic is a good candidate for future work.

An important distinction between caching [2] strategies is the grain at which data is cached. The granularity issue can be viewed in terms of three alternatives:

- *Object Grain:* Data is stored and evicted at the object grain. The advantage of this approach is that useful objects can be extracted from a page and stored in the cache, while less useful objects can be discarded. This leads to better cache utilisation, which is particularly important when the cache size is very small compared to the database size. However, the CPU and meta data overhead of maintaining data at such fine grain is high. Objects from a loaded page need to be copied one at a time into the object cache. The Thor system uses this approach [Liskov et al. 1996].

- *Page Grain:* Data is stored and evicted at the page grain. The main advantage of this approach is the low cost of buffer management. Low costs arise from the fact that pages are *fixed size* and the fact that *disk* to *memory* transfer is at the *page grain*. It is cheaper to manage *fixed sized* pages than it is to manage *variable sized* objects. Loading and caching data at the same grain (page grain) removes the need to perform memory copies. In object grained caching, objects need to be extracted and copied into the object cache upon loading. The disadvantage of this approach is that if objects are not well clustered, the cache can contain many useless objects, leading to high memory wastage. Systems that use page grained caching include: Platypus [He et al. 2000], EXODUS [Carey et al. 1986], O_2 [Deux 1991] and ObjectStore [Lamb et al. 1991].

- *Dual Grain:* In this approach main memory is divided into multiple buffers. Each buffer can be either page or object grained. This approach has the advantage that well clustered pages can be left in the page buffer and the badly clustered pages can have their objects copied into the object buffer. Kemper and Kossmann [1994] show that dual grained caching often outperforms page grained

[2] Throughout the rest of this thesis we use the word 'cache' to denote the main memory cache.

caching when using the OO7 benchmark [Carey et al. 1993]. However, they use naive clustering strategies that do not offer high quality clustering. We believe page grained caching will outperform dual grained caching when high quality clustering algorithms are used. Dual grain caching is also used in SHORE [Carey et al. 1994].

In this thesis we choose to explore *page grained caching*. The reason for this choice is three fold: the popularity of the page grained caching (ObjectStore [Lamb et al. 1991], EXODUS [Carey et al. 1986], O_2 [Deux 1991] and Platypus [He et al. 2000]); the good performance of page grained caching when the system is well clustered; and our aim of exploring the effects of clustering on OODBMS performance. We believe page grained caching outperforms both object and dual grained caching when the system is well clustered. This is because when the system is well clustered, the savings made on reduced cache maintenance costs outweigh any cache space wastage costs incurred.

2.4.4 Data Transfer Grain

The granularity of data transferred between client and server caches has a large effect on system performance. There are typically two grains at which OODBMSs transfer data:

- *Object server architecture:* In this architecture the unit of data transfer is groups of one or more objects. The advantage of this design is that only those objects that are needed are transferred to the client. The disadvantage is that the cost of transfer per object is high. In the case of transferring a single object, the network latency is incurred for just one object, thus making the transfer cost per object high. When a group of objects is transferred, the CPU cost of assembling the desired objects into one unit of transfer can place a high per object transfer cost on the server. This technique is employed by Thor [Liskov et al. 1996].

- *Page server architecture:* The server sends pages of data to the clients. This approach incurs a low per object cost since one instance of network latency is incurred for all the objects residing in the page or pages. In addition, no processing cost is incurred for grouping objects for sending (as needed by the object server architecture). However, when objects are not well clustered, many useless objects may be transferred to the client. Many object-oriented databases use this approach: EXODUS [Carey et al. 1986], ObjectStore [Lamb et al. 1991], O_2 [Deux 1991] and Platypus [He et al. 2000].

In this thesis we focus on a single stand-alone node of the peer-to-peer network model in which clients and servers share the same main memory cache. In such a model, data transfer between clients and server is done through the shared main memory cache and thus network latencies arising from either object grained or page grained data transfer are non-existent. However, the techniques developed in this thesis are best suited for use in page server architectures.

2.5 Conclusion

After reviewing the different OODBMS concepts and design alternatives, we identify our decision to explore issues surrounding storage management of stand-alone peer-to-peer OODBMSs using page grained caching. The stand-alone configuration allows us to ignore network issues, thus enabling us to focus on techniques that minimise the effects of disk IO. However, the techniques that are developed in this thesis can be extended to work in the general peer-to-peer multi-node network model (see section 7.2). We choose the peer-to-peer network model since it allows multiple clients and one server to exist on a single node of the system. Page grained caching is chosen for its popularity and its superior performance when the system is well clustered. In the next chapter we define the scope the thesis more vigorously by defining a precise system model and an integrated cost model.

Chapter 3

An Integrated Cost Model

The previous chapter defined key OODBMS concepts and design alternatives. In addition, it served as a means of defining the scope of the thesis within these design alternatives. In particular, the system architecture was loosely defined by choosing among the design alternatives.

This chapter further narrows the scope of the thesis by providing a more rigorous definition of the system model studied in the thesis. Secondly, this chapter provides an *integrated cost model* based on the system architecture. This cost model serves two purposes: it captures the performance implications of the concurrent interactions between the different buffer management techniques and the client threads; and it provides a precise means of defining problems studied in later chapters.

The chapter starts by describing the system and reference models used throughout the thesis. Using these models, we define an integrated cost model depicting how the four buffer management areas affect system performance. Finally, we outline more practical reference models which are used in the thesis.

3.1 System Model

In this section we model a multi-user database that allows concurrent execution of user applications. The model has the following system entities (see section 2.4 for a discussion of possible system architectures):

- A database server serving one or more client processes using one or more CPUs.
- A number of concurrently running client processes on the same machine as the server, running applications requesting objects from the server.
- A disk only accessible to the server.
- A main memory *page* cache that the clients and server share.
- A disk queue, that rearranges disk page requests.
- A dynamic clustering thread which periodically changes the object to page mapping.

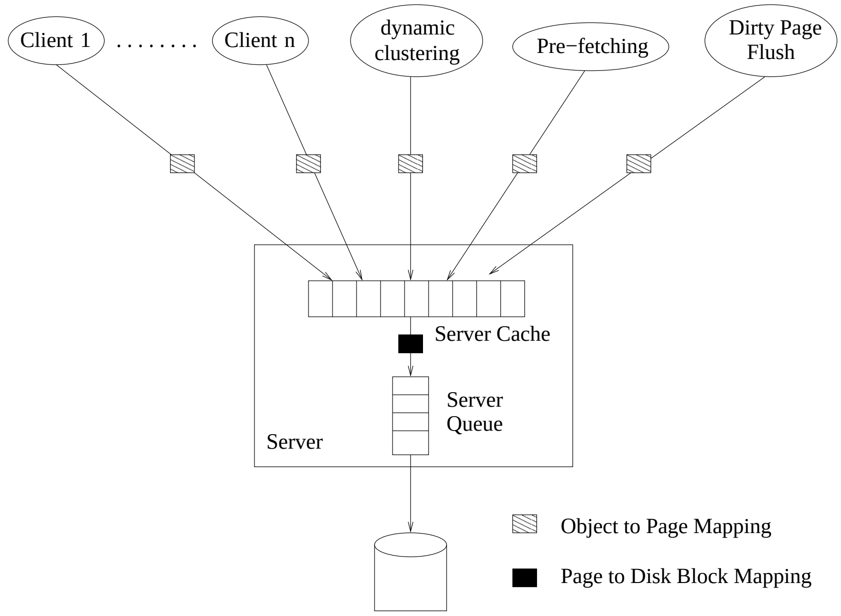


Figure 3.1: Diagram of system model. Circled entities are concurrently running threads or processes. All threads and processes are executed on the same machine.

- A prefetch thread which pre-loads pages from the disk into cache.

Figure 3.1 shows a diagram of the system model explored in this thesis. We chose to focus our attention on a single stand-alone node of a peer-to-peer system. The reason, as outlined in section 2.4, is that we would like to focus our analysis on the effects of disk IO.

In our model, clients make object requests which are mapped to page requests at the server via the object to page mapping. A client sending an object request is blocked until the requested object becomes available. The clients and server share the same main memory *page cache*. The cache holds at most k pages. If a requested page is found in the cache, it is immediately returned to the requesting thread or process; otherwise, the requested page is mapped to a disk block via the page to disk block mapping and placed on the server queue. The page to disk block mapping allows the server to locate the requested page on disk. The server queue rearranges disk page requests in a way that depends on the disk and the client scheduling policy. Finally, the rearranged requests coming out of the queue are sent to the disk. In order to keep the cost models simple we will not model the effects that the page to disk block mapping has on performance.

A fetch can occur as a result of a cache miss (demand fetch) or as a result of an anticipated miss (prefetch). Prefetches are initiated by the prefetch thread. No more than one fetch is allowed at any one time, that is, there is no concurrency at the disk IO level. Concurrency between read and write disk IO is also not permitted. When a page fetch is requested and the cache is full, an in cache page must be evicted. If the evicted page is clean, it is discarded, else it is written back into disk before the requested page is loaded. While the fetch is in progress, neither the incoming page nor the evicted page can be accessed. Disk write requests, like read requests, are also placed on the server queue.

Periodically, a dynamic clustering thread interrupts the program's execution and re-organises the database and thereby changes the object to page mapping. Each object is mapped to exactly one page.

3.2 Reference Model

In this section we describe the reference model used by the cost models of this thesis. An important characteristic of the reference model is that it allows the definition of multiple concurrent traces. In this thesis the reference model is only used to reflect sequences of *client* thread requests, it is not used for modeling *systems* thread requests.

Let T_n be the set of n independent concurrent traces of client program execution:

$$T_n \;=\; \{t_1, t_2, ..., t_n\} \tag{3.1}$$

Each trace t_i is composed of a sequence of h_i object requests:[1]

$$t_i \;=\; r_1, r_2, ...r_{h_i} \tag{3.2}$$

Let $x_i(T_n)$ be one possible interleaving of the client traces of T_n. There are many possible interleavings of T_n, each one generates a different global request sequence $G(x_i(T_n))$.

3.3 Problem Statement

In this section we use the reference model of section 3.2 to define the general problem that the thesis addresses.

Given a set of n independent client thread traces T_n, we define the best integrated buffer management technique as one that produces the minimum average execution time $\overline{P}(T_n)$:

$$\mathrm{Min}(\overline{P}(T_n)) \;=\; \mathrm{Min}(\frac{\sum_{i=0}^{NI} P(x_i(T_n))}{NI}) \tag{3.3}$$

NI is the number of possible interleavings. $P(x_i(T_n))$ is the average of the execution time of all of the traces of T_n under the $x_i(T_n)$ interleaving:

[1]In this reference model we do not include the concept of time, we merely specify a sequence of object requests. The cost model of section 3.4 adds the time dimension absent in this model.

$$P(x_i(T_n)) = \frac{\sum_{j=0}^{n} ET(x_i(T_n),t_j)}{n} \tag{3.4}$$

Where $ET(x_i(T_n),t_j)$ is the execution time of the t_j trace under the $x_i(T_n)$ interleaving. n is the total number of traces.

The average execution time under all possible client thread interleavings is the metric to be optimised. This is because in general client threads can start at any time and many system and user factors can affect the interleaving generated. Therefore the buffer management technique developed should perform well for an average of all possible interleavings.

3.4 An Integrated Cost Model

This section describes an integrated cost model that incorporates the effects that static clustering, dynamic clustering, buffer replacement and prefetching have on system performance. This cost model is defined using the reference model of section 3.2.

The threads modeled in this section include:

- This client thread (TC)
- Other client threads (OC)
- A dynamic clustering thread (DC)
- A prefetcher thread (P)

Equation 3.5 depicts the execution time for the trace $t_i = (r_1, r_2, \ldots\ldots, r_h)$ under the $x_i(T_n)$ interleaving.

$$\begin{aligned} ET(x_i(T_n),t_i) = & \sum_{r=0}^{h} (IO_{TCR}(r) + CPU_{TC}(r)) + \\ & \sum_{r=0}^{h} (IO_{DCR}(r) + CPU_{DC}(r)) + \\ & \sum_{r=0}^{h} (IO_{PIR}(r) + CPU_{P}(r)) + \\ & \sum_{r=0}^{h} (IO_{OT}(r) + CPU_{OT}(r)) \end{aligned} \tag{3.5}$$

The following is an explanation of the terms of equation 3.5:

- The $IO_{TCR}(r)$ and $CPU_{TC}(r)$ terms refer to the disk IO and CPU resources consumed by *this* client to process the trace:

$IO_{TCR}(r)$: time spent by this client waiting for its own requested object to be read from disk between references r and $r+1$. Suppose the requested object resides on page p, then $IO_{TCR}(r)$ equals the portion of the time that loading page p has caused this client to be blocked. It does not include the time spent waiting for the load completion of other pages on the server queue. In the case that the requested page is already in the process of being loaded by another thread, then $IO_{TCR}(r)$ equals the remaining load time. The value of this term is affected by *static clustering*, *dynamic clustering*, *buffer replacement* and *prefetching*. The primary objective of these buffer management techniques is to reduce the value of this term.

$CPU_{TC}(r)$: time spent on the CPU by this client between references r and $r+1$.

- The $IO_{DCR}(r)$ and $CPU_{DC}(r)$ terms refer to the time this client spent blocked due to disk IO and CPU usage by the *dynamic clustering thread*:

$IO_{DCR}(r)$: time this client was blocked due to disk read IO activity caused by the dynamic clustering thread between references r and $r+1$.

$CPU_{DC}(r)$: time this client was blocked due to CPU activity caused by the dynamic clustering thread between references r and $r+1$.

- The $IO_{PIR}(r)$ and $CPU_P(r)$ terms refer to the time this client spent blocked due to incorrect prefetch disk IO and prefetch CPU usage by the *prefetch thread*:

$IO_{PIR}(r)$: time this client was blocked due to incorrect prefetch IO by the prefetcher thread between references r and $r+1$. Incorrect prefetch IO is defined as a prefetched page not corresponding to the next disk page request. This definition of $IO_{PIR}(r)$ does not incorporate the benefits of a prefetched page being referenced after the next disk page load but before it is evicted (section 6.3.3 addresses this issue). The reason for using this simplistic definition of $IO_{PIR}(r)$ is that this section is only intended to give the reader a broad overview of the way buffer management techniques can affect system performance. We leave more in-depth analysis to the later chapters.

$CPU_P(r)$: time this client was blocked due to CPU activity caused by the prefetcher thread between references r and $r+1$.

- The $IO_{OT}(r)$ and $CPU_{OT}(r)$ terms refer to the time this client spent blocked due to disk IO and CPU usage by other sources including *other clients* and *buffer management*:

  - Other disk IO:

$$IO_{OT}(r) \;=\; IO_{OCR}(r) + IO_{BW}(r) \tag{3.6}$$

$IO_{OCR}(r)$: time this client spends blocked due to read IO done by other clients between references r and $r+1$. *Static clustering*, *dynamic clustering*, *buffer replacement* and *prefetching* affect this term in the same way as $IO_{TCR}(r)$.

$IO_{BW}(r)$: time this client spends blocked due to write IO initiated by the *buffer replacement algorithm* between references r and $r+1$. In addition to buffer replacement, this term is also affected by both *static* and *dynamic clustering*. Static and dynamic clustering defines the object to page mapping which in turn determines which pages are dirtied by user applications when they dirty objects. Dynamic clustering also affect $IO_{BW}(r)$ by dirtying pages it chooses for dynamic reorganisation.

– Other CPU:

$$CPU_{OT}(r) \;=\; CPU_{OC}(r) \tag{3.7}$$

$CPU_{OC}(r)$: time this client spends blocked due to CPU activity caused by the other clients between references r and $r+1$.

3.5 Practical Reference Models

The object level reference trace described in section 3.2 offers very accurate information for use in the cost models. However, in practice the trace is too expensive to collect. Therefore in practice, standard statistical models based on stochastic processes are used. These statistical models capture average program behavior and use it to predict the probabilities of certain events occurring in the future. All buffer management algorithms proposed in this thesis use one or more of the reference models described in this section.

In this section we firstly describe each reference model for the single thread case. Later, in section 3.5.4 we describe how these practical reference models can be used in the concurrent multi-threaded case.

We now define some terms that will be used in the remainder of the section. Let S represent the set of N objects representing the entire reference trace object population:

$$S \;=\; \{1,2,3,.......,N\} \tag{3.8}$$

Let R_n represent the object level reference trace:

$$R_n \;=\; r_1,r_2,....,r_n \tag{3.9}$$

3.5.1 Independent Reference Model

The independent references model, also known as IID (sequence of Independent and Identically Distributed random variables), describes the reference trace as a random process. It is a simple reference model which considers references occurring in the reference trace to be random independent events. At any time t, the probability that an object x appears in the trace is fixed and only depends on x:

$$\pi(x) \equiv Prob\{R_t = x\} \tag{3.10}$$

where:

$$\sum_{x=1}^{N} \pi(x) = 1 \tag{3.11}$$

The IID model can be expressed as a N row vector of probabilities:

$$\vec{\pi} \equiv [\pi(1)\ \pi(2)\ ...\ \pi(N)]^T \tag{3.12}$$

To estimate the IID model from the reference trace R_n, we use the following unbiased estimator for each vector element [Arnold 1978]:

$$\hat{\pi}(x) = \frac{\sum_{t=1}^{n} \delta_{xR_t}}{n} \tag{3.13}$$

where:

$$\delta_{xy} = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$$

The statistics for the IID model can be collected very easily, and impose very small overheads. To collect the statistics required to predict $\pi(x)$, one merely needs to count the number of times the object x has been referenced. The storage cost of IID is $O(N)$, where N is the number of objects in the trace.

3.5.2 Simple Markov Chain Model

The simple Markov chain model (SMC) describes the reference trace as a Markovian process. A simple markov chain is a random sequence in which the dependency goes back only one unit in time. In other words, the future probabilistic behavior of the process depends only on the present state of the process and is not influenced by its past history. In the case of our reference trace, at any time t the probability that an object y appears in the trace is fixed, and only depends on y and the previously requested object x:

$$P(x,y) \equiv Prob\{R_t = y | R_{t-1} = x\} \tag{3.14}$$

where:

$$\sum_{y=1}^{N} P(x,y) = 1 \tag{3.15}$$

The SMC model can be expressed as a $N \times N$ matrix of conditional probabilities:

$$P \equiv \begin{bmatrix} P(1,1) & P(2,1) & .... & P(N,1) \\ P(1,2) & P(2,2) & .... & P(N,2) \\ & & .... & \\ P(1,N) & P(2,N) & .... & P(N,N) \end{bmatrix} \tag{3.16}$$

To estimate the SMC model from the reference trace R_n we can use the following unbiased estimator for each matrix element [Arnold 1978]:

$$\hat{P}(x,y) = \frac{\sum_{t=2}^{n} \delta_{xR_{t-1}} \delta_{yR_t}}{\sum_{t=1}^{n-1} \delta_{xR_t}} \tag{3.17}$$

The SMC matrix P can also be represented as a labeled directed graph $G(P)$: the vertices represent the objects; the edges are labeled with elements of P. That is, an edge from node x to node y is labeled by $\hat{P}(x,y)$. In this case, the reference trace is considered as the result of a random walk on graph $G(P)$. $G(P)$ expresses the "average" behavior of the program.

The storage cost of the SMC model is much greater than the IID model, since the SMC model stores the frequency by which any pair of objects are accessed consecutively. The storage cost of SMC is $O(E)$, where E is the number of edges in $G(P)$. Whereas, the storage cost of IID is $O(N)$, where N is the number of objects in $G(P)$.

3.5.3 Higher Order Markov Chains model

The higher order Markov Chains model (HMC) model is a generalisation of the SMC model to a k-th order stochastic process. The future probabilistic behavior of a process depends on the past k states. In the case of the reference trace, at any time t, the probability that an object y appears in the trace is fixed, and only depends on y and the k most recently requested objects. Although this model is more accurate, it involves storing information for sequences of k object references. The large statistics storage requirement of this model makes it impractical to apply at the object grain. However, at the page grain, HMC is more practical. Some of the prefetching algorithms studied in chapter 6 use this model at the page grain.

3.5.4 Concurrency

Up to now the practical reference models are described for the single threaded case. This section details the issues surrounding the use of the practical reference models in a concurrent multi-threaded environment.

3.5.4.1 Independent Reference Model

The independent reference model (IID) considers references occurring in the reference trace to be random independent events. Therefore IID does not model serial dependency. Hence *any* interleaving of N concurrent thread executions produces the same resultant global IID model. This suits the purposes of this thesis very well. The reason is that IID is used in this thesis to model global system behaviour for making optimisation decisions that benefit *average program* behaviour (average performance of all possible interleavings). This concurs with the problem statement in section 3.3 namely, we are interested in finding buffer management techniques that give the best average performance of all possible thread interleavings.

3.5.4.2 Markov Chain Models

Unlike IID, Markov chain models (MCM), capture the serial dependency between references. Naively applying MCM to the global reference stream (produced from one particular interleaving) results in producing a MCM model that only reflects the characteristics of that particular interleaving. This naive application of MCM is not desirable since we use MCM to model average programming behaviour which should reflect an average of all possible interleavings (see the problem statement in section 3.3). In contrast, applying MCM to the trace generated by each thread separately and then aggregating the individual models to produce one global model results in the same model regardless of the interleaving. This approach captures the serial dependencies between references occurring in the same thread trace which is independent of the particular interleaving. The algorithms that use MCM in this thesis are designed to give best average program performance (average of all possible interleavings).

3.6 Conclusion

This chapter defined the system models that will be the subject of study throughout the rest of this thesis. In addition, the reference model used by the cost models of this thesis was also described. An integrated cost model was described. The integrated cost model serves as a general mechanism for outlining opportunities for improving buffer management techniques. The next chapter outlines techniques for optimising the integrated cost model by exploiting the synergies between *static clustering* and *dynamic clustering*.

Chapter 4

Opportunistic Prioritised Clustering Framework (OPCF)

The previous three chapters explained the general problem that this thesis addresses. This was done by defining the scope of this thesis along with an integrated cost model that identified how the different buffer management areas affect system performance.

As explained in section 1.4, we do not focus on finding an integrated solution that incorporates synergies between all four buffer management areas, but instead find solutions that incorporate synergies between pairs of buffer management techniques. In this chapter we explore the advantages of developing buffer management techniques that incorporate synergy between *static clustering* and *dynamic clustering*. This is done by developing the opportunistic prioritised clustering framework (OPCF). The framework transforms existing static clustering algorithms into dynamic algorithms.

First, the integrated cost model of section 3.4 is used to define the problem we are addressing in this chapter. Second, the integrated cost model is used to explain how the different components of OPCF improve performance. Third, we explain how OPCF is used to develop two new dynamic clustering algorithms. Finally, the results of a simulation study comparing the performance of the two new algorithms with existing highly competitive algorithms is presented.

4.1 Introduction

Ever since the 'early days' of database management systems, clustering has proven to be one of the most effective performance enhancement techniques [Gerlhof et al. 1996]. This is because the majority of object accesses in an object oriented database are navigational. Consequently, related objects are often accessed consecutively. Clustering objects in an object oriented database reduces disk IO by grouping related objects onto the same disk page. 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 organisation of objects on disk to more closely reflect the prevailing pattern of object access.

The majority of existing clustering algorithms are static [Tsangaris 1992; Banerjee et 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 24 hour database access. In contrast, dynamic clustering algorithms re-cluster the database while database applications are in operation. Applications that require 24 hour database access and involve frequent changes to data access patterns may benefit from the use of dynamic clustering.

In our view, three properties are 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 minimise the IO footprint for re-organisation; and
- A prioritisation of re-clustering so the worst clustered pages are re-clustered first.

Despite the great body of work 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 chapter we address this omission by developing a dynamic clustering framework 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 model in section 3.4, the clustering process itself may generate IO, loading data pages for the sole purpose of object base re-organisation. However, most researchers have chosen to ignore these sources of IO generation and instead concentrate on developing the dynamic clustering algorithm that minimises the number of transaction read IOs generated by the client processes. In this chapter we address this omission by incorporating opportunism into our framework. Opportunism eliminates clustering read IO by choosing in-memory pages for re-clustering.

The disruptive nature of re-clustering dictates that dynamic clustering algorithms must be incremental. This means that only a small portion of the object base can be re-clustered in each iteration. We believe that it should be the worst clustered pages that are re-clustered first. We term this property *prioritisation*. Dynamic clustering algorithms produced by our framework all possess the prioritisation property.

4.2 Related Work

The re-organisation phase of dynamic clustering can incur significant overhead. Two of the key overheads are increased write contention,[1] and IO.

To reduce write contention, most dynamic clustering algorithms are designed to be incremental and thus limit the scope of re-organisation. However, *DROD* [Wietrzyk and Orgun 1999] is the only algorithm that we are aware of that limits the scope of reorganisation so that only in-memory objects are re-clustered. Wietrzyk and

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

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 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 statisitical 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 at 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-organisation. 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-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 [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-organisation iteration by breaking the re-organisation workload into re-clustering units and only re-organisating one unit in each iteration.

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

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. In this chapter, two existing static clustering algorithms are transformed into dynamic clustering algorithms using OPCF and compared to two existing dynamic clustering algorithms, DSTC [Bullat and Schneider 1996] and DRO [Darmont et al. 2000].

4.3 Preliminaries

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

4.3.1 Problem Definition

Using the integrated cost model of section 3.4 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 t_i, an initial object to page mapping (initial clustering), a buffer replacement algorithm and an interleaving $x_i(T_n)$, we seek the *dynamic clustering algorithm* that minimises the execution time $ET(x_i(T_n),t_i)$ of t_i under $x_i(T_n)$ using equation 3.5 of section 3.4. This is formulated as:

$$\mathrm{Min}(ET(x_i(T_n),t_i)) = \mathrm{Min}(\sum_{r=0}^{h} IO_{TCR}(r) + \sum_{r=0}^{h}(IO_{DCR}(r) + CPU_{DC}(r)) + \sum_{r=0}^{h} IO_{OT}(r)) \quad (4.1)$$

The $CPU_{TC}(r)$ and $CPU_{OT}(r)$ terms from equation 3.5 have been omitted due to our focus in this chapter on finding the best dynamic clustering algorithm. 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.

As described in section 3.4 the $IO_{OT}(r)$ term of equation 4.1 can be further decomposed into:

$$IO_{OT}(r) \ = \ IO_{OCR}(r) + IO_{BW}(r) \tag{4.2}$$

We do not aim to produce dynamic clustering algorithms that reduce $IO_{BW}(r)$ by re-organising 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.

4.3.2 Constraints

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

4.3.2.1 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 T_c 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:

$$\sum_{r=i}^{j} (IO_{DCR}(r) + CPU_{DC}(r)) \ < \ T_c \tag{4.3}$$

4.3.2.2 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 T_t 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:

$$\sum_{r=i}^{j} (IO_{TCR}(r) + IO_{OT}(r) + CPU_{OT}(r)) \ > \ T_t \tag{4.4}$$

Constraints 4.3 and 4.4 combine to limit the frequency and duration of re-clustering iterations.

4.3.3 Assumptions

The work in this chapter 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. However, the techniques in this chapter can still be applied when large objects are present. For example, large objects can be placed in a separate area of the object store and dynamic clustering algorithms can ignore them.

3. The patterns of object access after and before each re-organising iteration bare some degree of similarity.

4.4 Opportunistic Prioritised Clustering Framework (OPCF)

In this section we outline in detail the main contribution of this chapter, 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.

4.4.1 Opportunism

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

$$\mathrm{Min}\left(\sum_{r=0}^{h} IO_{DCR}(r)\right) \tag{4.5}$$

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

4.4.2 Incrementality

OPCF limits the disruption caused by the dynamic clustering thread by *incrementally* re-organising the database. This property of OPCF allows the dynamic clustering algorithm to meet the constraints of equations 4.3 and 4.4. OPCF achieves constraint 4.3 by placing a fixed bound on the number of pages re-clustered in each re-clustering iteration. Constraint 4.4 is accomplished by allowing users to control the frequency with which re-clustering is triggered.

[3] Objects larger than one page in size.

4.4.3 Prioritisation

Incrementality specifies that re-organisation should be partitioned and only one portion of the database should be re-organised in each iteration. Prioritisation specifies that the worst clustered portion should be targeted for re-organisation in each iteration. We now explain the aim of prioritisation by using equation 4.1. Prioritisation aims to achieve large reductions of $IO_{TCR}(r)$ and $IO_{OCR}(r)$ costs while incurring only small $IO_{DCR}(r)$ and $CPU_{DC}(r)$ costs. 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.

4.4.4 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-organisation 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-organisation the algorithm must be able to operate within a limited scope.

- *Define Clustering Badness Metric*: OPCF prioritises 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 hot[4] 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 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.

[4]By hot objects we mean objects accessed more frequently.
[5]Cluster analysis simply refers to calculating clustering badness of pages of the store.

- *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 must be chosen to form the scope of re-organisation. 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.[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).

4.5 Two Example 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.

4.5.1 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 M frame requests, eliminating duplicates and computing the cardinality of the resulting set. Therefore, the larger the cardinality, the fewer the duplicates, 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.

[6] In contrast, when re-organisation occurs at a cluster grain, each re-organisation can be more strongly targeted towards a particular cluster or clusters, and so is more likely to identify larger clusters.

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 thesis to serve as a tool for discussing the relative merits of existing static clustering algorithms.

4.5.2 Static Probability Ranking Principle (PRP)

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 graph[7] 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.

4.5.3 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.

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

- *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 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:

$$CB(p) = (\ \sum_{i \in p} heat_i\ ) \ \times \ (\ \sum_{i \in p} (size_i / heat_i)\ ) \tag{4.6}$$

  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-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 4.1 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.

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 reorganisation. The time complexity is determined by the sorting of objects within the scope of re-organisation.

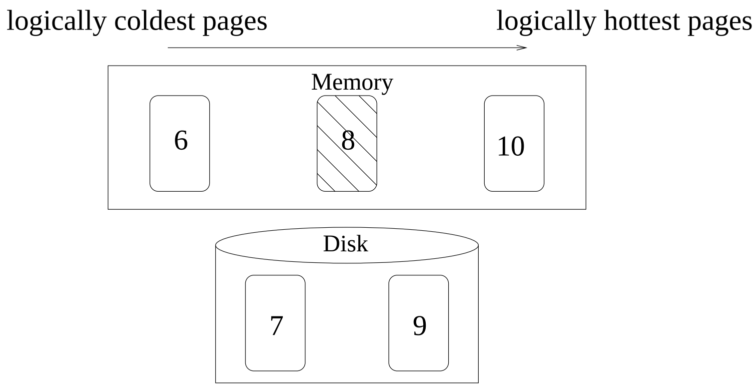


Figure 4.1: In this example the currently worst clustered page is 8, so the scope of reorganisation for opportunistic dynamic PRP is pages 6, 8 and 10 (where page numbers reflect heat-order). If opportunism is not used, the scope would be pages 7, 8 and 9.

4.5.4 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 ($\hat{P}(x,y)$) of the SMC metric (see section 3.5.2, page 22). 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) [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.

4.5.5 Dynamic Graph Partitioning

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

- *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.

- *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:

$$CB(p) = \Big(\sum_{i \in p} heat_i \Big) \times \Big(\sum_{i \in p} external\ tension_i \Big) \tag{4.7}$$

  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-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 4.2 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 4.5.2).

The particular graph partitioning algorithm implemented for the results section of this chapter is the greedy graph partitioning (GGP) algorithm [Gerlhof et al. 1993].

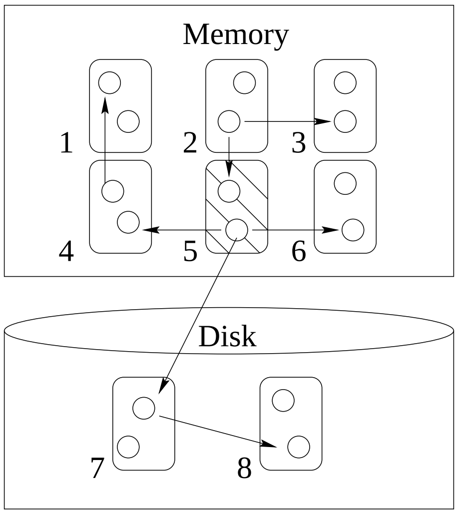


Figure 4.2: In this example the worst clustered page is 5 and the scope of re-organisation for opportunistic dynamic graph partitioning are pages 2, 4, 5 and 6. The scope of re-organisation for non-opportunistic dynamic graph partitioning are pages 2, 4, 5, 6 and 7.

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 $O(e_s\ log\ e_s)$.

4.6 Two Existing Dynamic Clustering Algorithms

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

4.6.1 Dynamic Statistical and Tunable Clustering (DSTC)

DTSC 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 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.

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.

4.6.2 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, 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).

4.7 Experimental Setup

In this section we describe experiments designed to compare the performance of algorithms produced using OPCF with two existing state of the art dynamic clustering algorithms, DSTC and DRO.

4.7.1 Simulation Environment

The experiments are conducted using the virtual object oriented database simulator, VOODB [Darmont and Schneider 1999]. VOODB is based on a generic discrete-event

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 4.1: VOODB parameters used.

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 IO subsystem. The correctness of VOODB has been validated for two real-world OODBs, O_2 [Deux 1991] and Texas [Singhal et al. 1992].

VOODB is implemented on top of the discrete-event simulation package for C++ (DESP-C++) [Darmont 2000]. DESP-C++ is a validated simulation package which performs 20 to 1000 times faster than competing simulation packages like the 'queuing network analysis package 2^{nd} generation' (QNAP2). The high performance of DESP-C++ allows us to test more complex workloads and system settings.

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 IO (the dominating metrics that determine the performance of dynamic clustering algorithms).

It is important to note that in terms of write IO, 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 thesis 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 4.1. The 'centralised' system class refers to the stand-alone system configuraton, 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.

4.7.2 Performance Evaluation Environment

We evaluate the performance of the dynamic clustering algorithms in two sets of experiments. First, we use the object clustering benchmark (OCB) [Darmont et al. 1998] to compare the performance of the dynamic clustering algorithms under stationary access pattern situations. Second, the dynamic object evaluation framework (DOEF) [He and Darmont 2003] is used to test the effects of changes in access patterns.

In this thesis we use traces generated from synthetic benchmarks instead of real world applications. This is due to two reasons: (1) the unavailability of real application traces in the OODBMS field, and (2) the ability to control benchmark parameters to generate multiple traces for detailed algorithm comparison. Firstly, in the area of OODBMS performance optimization, no existing work have managed to use real application traces. This is mainly due to privacy issues and the overheads in keeping a running trace of OODBMS object accesses.

Secondly, a benefit of synthetic benchmarks is the ability 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 in a particular way.

However, it has been noted in Wilson, Johnstone, Neely, and Boles [1995] that the synthetic benchmarks may not produce traces that are indicative of real world applications. The work of [Wilson et al. 1995] 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.

4.7.2.1 Stationary Access Pattern Evaluation Environment

In this thesis the OCB benchmark [Darmont et al. 1998] 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 experiments in this chapter.

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 experiments reported in this chapter, 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 4.2 (a).

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

(a) OCB database 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

(b) OCB workload parameters

Table 4.2: Parameters used for OCB.

The OCB workload used in this study included simple, hierarchical and stochastic traversals [Darmont et al. 1998]. 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 [Tsangaris and Naughton 1992]. Each transaction involved the execution of one of the three traversals. The OCB workload parameters used are shown in table 4.2 (b).

We introduce skew into the way traversal roots are selected. Roots are partitioned into hot and cold regions. In all experiments the hot region is set to 3% of the size of database and has an 80% probability of access.[8] These settings are chosen to represent typical database application behaviour. Gray and Putzolu [1987] cites statistics from a real videotext application in which 3% of the records got 80% of the references. Carey, Franklin, Livny, and Shekita [1991] use a hot region size of 4% with a 80% probability of being referenced in the HOTCOLD workload have been used to measure data caching trade-offs in client/server OODBMSs. Franklin, Carey, and Livny [1993] 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.

4.7.2.2 Dynamic Access Pattern Evaluation Environment

In this section we describe the dynamic object evaluation framework (DOEF) [He and Darmont 2003]. 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 accomplished by varying the way traversal roots of OCB are selected. DOEF divides traversal roots[9] 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 chapter. 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 experiments we use the same OCB database parameters as those shown in table 4.2 (a). However, we use different workload parameters. The operation we have used for all of the experiments is the simple, depth-first traversal with traversal depth 2. The simple traversal is chosen since it is the only traversal that

[8] That is, there is a 80% probability that the root of a traversal is from the hot region.
[9] All database objects can be used as traversal roots.

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 4.3: Parameters for DOEF.

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 experiment involved executing 10,000 transactions.

The DOEF parameters used in this chapter are shown in table 4.3. 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.

4.7.3 Dynamic Clustering Algorithms Tested

We compare the performance of four dynamic clustering algorithms in this chapter. 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 4.4. In order to tune the clustering algorithms we tested a range of different parameter settings for each algorithm in each experiment. Then for each algorithm we used the set of parameters that gave the best overall result for all the experiments. We found algorithm performance was not very sensitive to parameter settings across different experiments. This means the same set of parameters usually performed well for all experiments or none of the experiments. For a more detailed description of the DSTC and DRO parameters please refer to [Bullat and Schneider 1996] and [Darmont et al. 2000] respectively. Detailed descriptions of OPCF algorithm parameters can be found in sections 4.4.4 and 4.5.5.

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

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

(a) DSTC 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

(b) DRO 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

(c) OPCF parameters

Table 4.4: Parameters used for the clustering algorithms DSTC, DRO, OP-PRP, and OP-GP.

- **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.

4.7.4 Evaluation Metric

All the results presented in this chapter are in terms of total IO. Total IO is the sum of transaction read IO[10], clustering read IO and clustering write IO. Thus, the results give an overall indication of the performance of each clustering algorithm, including each algorithm's clustering IO overheads.

4.8 Experimental Results

In this section we report the results of experiments 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).

4.8.1 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 4.3. 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 IO than DRO and 48% less IO than DSTC. The performance advantage can be attributed to OP-GP's opportunistic behaviour. OP-GP reduces many clustering read and write IOs 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 IO is less likely to be absorbed by user applications. Second, at small buffer sizes, read IO 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.

[10] Recall in section 4.3.1, Equation 4.2 that we are only interested in improving transaction read performance.

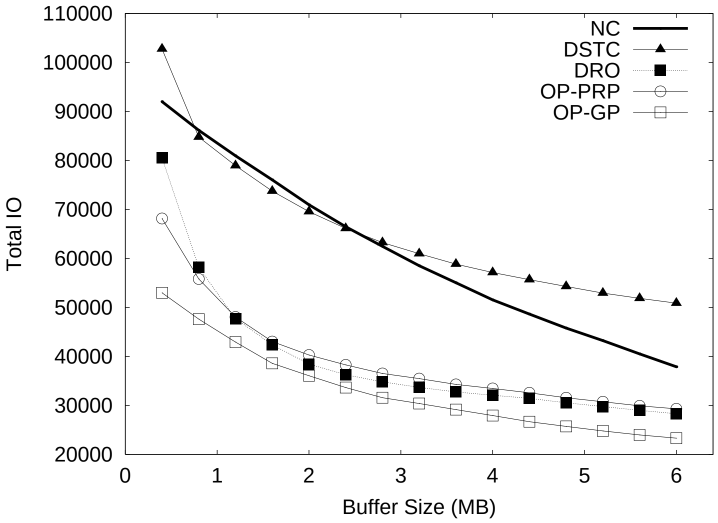


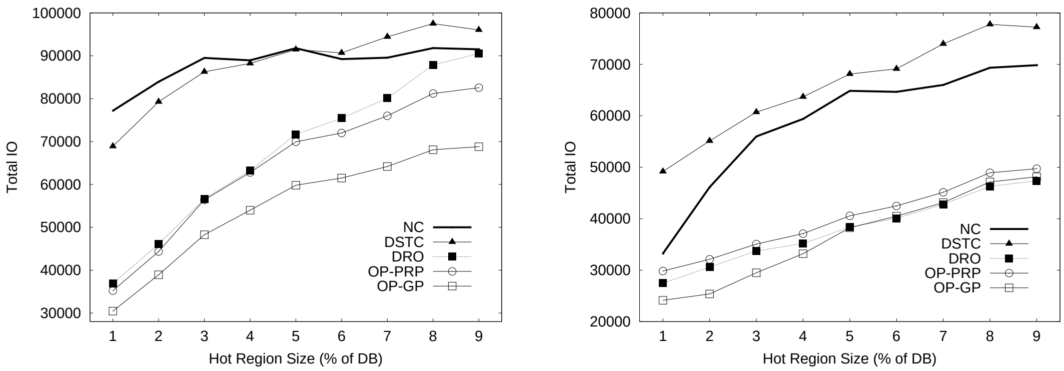
Figure 4.3: Varying buffer size.

At large buffer sizes (above 1MB) OP-GP maintains its lead over DRO at between 5% and 15% less IO produced. 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 IO. Second, DSTC's tuning parameters lack flexibility. In our experiments, 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 tested extensively with different parameters settings for all the algorithms and only used the settings that gave the best overall performance.

4.8.2 Varying Hot Region Size

In this section we examine the effect of changing hot region size on the dynamic clustering performance. The hot region probability of access is set to 80% for the reasons explained in section 4.7.2.1. The results for two buffer size settings of 1MB and 4MB are reported in figures 4.4 (a) and (b), respectively. The results in this experiment 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.



(a) 1MB buffer size.

(b) 4MB buffer size.

Figure 4.4: Varying hot region size.

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 a high 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 IO as a larger portion the working set is disk resident. DRO also performs more clustering write IOs 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 IO. In contrast, OP-GP's opportunistic behaviour combined with its *bounded* scope of re-organisation results in a smaller clustering IO footprint, for both small and large hot region sizes.

At the larger buffer size of 4MB (figure 4.4 (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 IO is required. Clustering write IOs are also decreased due to fewer dirty page evictions.

4.8.3 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 4.7.2.1. The results of two buffer size settings of 1MB and 4MB are reported on figure 4.5 (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 IO overheads for marginal transaction read IO 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 IO 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 IO resources re-clustering the cold region.

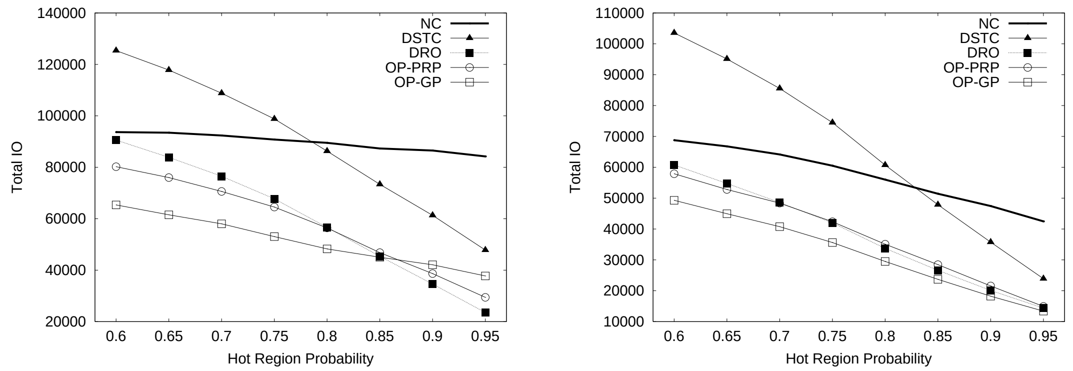
The results for the 4MB buffer size (see figure 4.5 (b)) again show OP-GP offering best performance when hot region access probability is low. When hot region access probability is high, OP-GP and DRO perform approximately the same. The reason for OP-GP's superior performance at low hot region access 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 (the cold region rarely gets touched and thus is rarely re-clustered). This fact means DSTC's clustering IO overheads rapidly diminish as the hot region access probability increases.

4.8.4 Moving Window of Change

In this experiment, 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 4.3. In this experiment 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 4.6 (a) and (b), respectively.

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



(a) 1MB buffer size.

(b) 4MB buffer size.

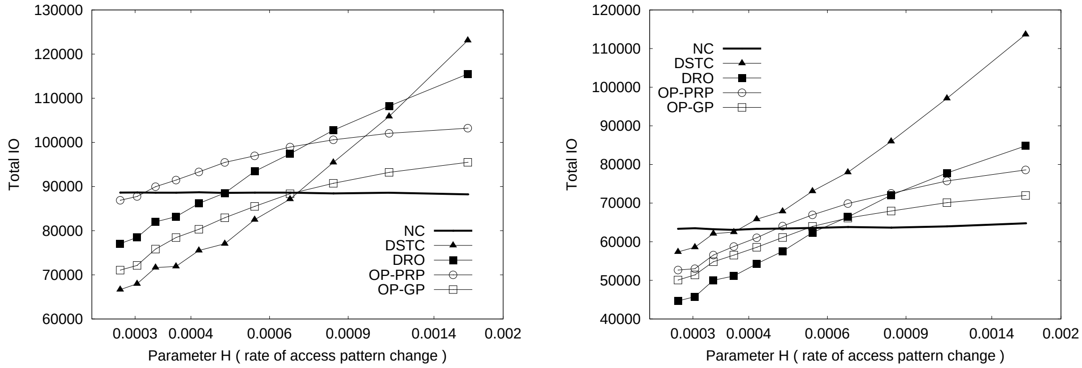
Figure 4.5: Varying hot region probability.

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 a high rate of access pattern changes.

4.8.5 Gradual Moving Window of Change Experiment

In this experiment, we used a less vigorous style of access pattern change, the *gradual moving window of change* protocol. Unlike the previous experiment, the hot region cools gradually instead of suddenly. The DOEF settings used are again shown in table 4.3. In this experiment 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 4.7 (a) and (b), respectively.

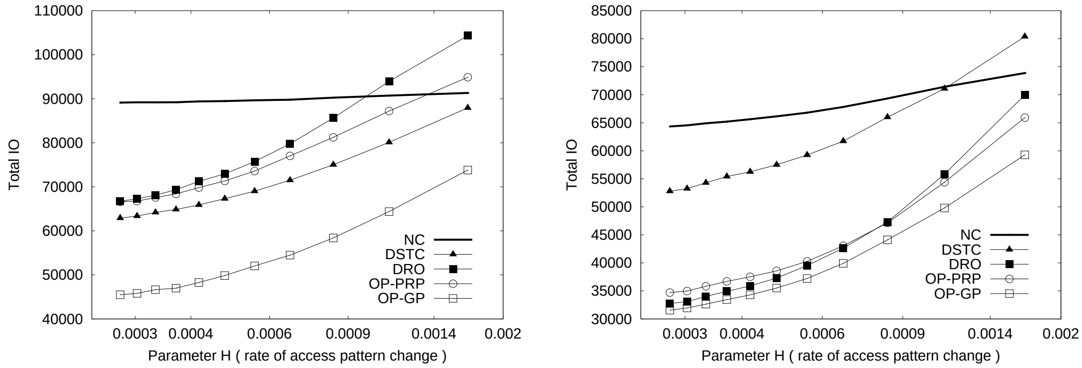
The results show OP-GP consistently outperforming the other dynamic clustering algorithm when this gradual style of change is used. This is due to OP-GP's policy of not deleting old statistics, which is beneficial instead of detrimental to clustering quality in this experiment. The gradual cooling of 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.

Figure 4.6: Moving window of change. The x-axis is in log_2 scale.



(a) 1MB buffer size.

(b) 4MB buffer size.

Figure 4.7: Gradual moving window of change. The x-axis is in log_2 scale.

4.8.6 Discussion

The results reported in this chapter 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 IO overhead, while sequence tension allows OP-GP to achieve high clustering quality. The combination of low clustering IO 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.

4.9 Conclusion

In this chapter we highlight the advantages of developing dynamic clustering algorithms that incorporate the synergies between *dynamic clustering* and *static clustering* algorithms. To this end we develop the opportunistic prioritised clustering framework (OPCF). OPCF is a general framework that creates dynamic clustering algorithms by placing opportunism and prioritisation into existing static clustering algorithms. In addition, application of the framework is straightforward and yet it produces robust dynamic clustering algorithms which outperform the existing highly competitive dynamic clustering algorithms DSTC and DRO in a variety of situations. The simplicity of the framework and the robustness of algorithms produced by it makes OPCF algorithms ideal candidates for inclusion in real OODBMS systems where workload conditions are likely to change with time.

The next chapter presents an exploration of the advantages of incorporating synergies between a different pair of buffer management techniques, namely *static clustering* and *buffer replacement*.

Chapter 5

Cache Conscious Clustering (C3)

The previous chapter explored the advantages of developing algorithms that incorporate the synergies between static and dynamic clustering algorithms. This was done by creating a general and simple framework in which existing static clustering algorithms can be transformed into dynamic clustering algorithms.

This chapter takes the same approach as the previous chapter, namely it presents a general and simple framework in which new synergistic buffer management techniques can be developed. However, the particular buffer management areas considered in this chapter differ. In this chapter we develop a framework which incorporates the synergy between *static clustering* and *buffer replacement*. The framework creates new static clustering algorithms which are cache conscious — aware of which types of pages are likely to be kept in memory by the buffer replacement algorithm.

First, this chapter uses the integrated cost model of section 3.4 to define the problem. Second, we refine the problem definition using the working set size (WSS) metric. Third, a new family of static clustering algorithms that minimises the WSS metric is defined by the establishment of the cache conscious clustering framework. Fourth, a new concrete static clustering algorithm produced from the framework is presented. Finally, the experimental results are reported.

5.1 Introduction

Static clustering and buffer replacement are two well established ways of reducing IO in OODBMSs. Traditionally static clustering algorithms have generally been designed to place objects likely to be co-referenced into the same page [Tsangaris 1992; Banerjee et al. 1988; Gerlhof et al. 1993; Drew et al. 1990]. On the surface this seems like a reasonable approach, since it confines object graph traversals as much as possible to one page. By minimising the likelihood of traversing out of the current page, the chances of requiring a disk load are minimised. However, upon closer inspection we can criticise this approach as being too conservative. This is because the assumption that navigations out of the current page have a high probability of causing a page load is only valid when either the cache size is one page or the cache size is larger but the buffer replacement algorithm only keeps pages cache resident for very short durations. In this chapter we exploit our knowledge of buffer replacement algorithm behaviour to design static clustering algorithms that tolerate navigations out of the current page so long as the navigation proceeds into another cache resident page. We term this new approach cache conscious clustering (C3). In order to make our approach more generally applicable we created the C3 framework. The C3 framework produces a *family* of static clustering algorithms that all possess the property of cache consciousness. Experimental results show a C3 algorithm called C3-GP outperforms existing static clustering algorithms in a variety of situations.

5.2 Related Work

Although there is much existing literature on both static clustering [Tsangaris 1992; Banerjee et al. 1988; Gerlhof et al. 1993; Drew et al. 1990] and buffer replacement [Nicola et al. 1992; O'Neil et al. 1993; Belady 1966; Johnson and Shasha 1994; Robinson and Devarakonda 1990; Lee et al. 1999], to our knowledge no prior work explores the synergies between the two techniques. In this section we first review existing clustering algorithms. Second, existing buffer replacement algorithms are reviewed.

5.2.1 Existing Static Clustering Algorithms

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

- *Object Graph (OG):* In this approach the object graph itself is used as the sole source of information for clustering purposes. The object graph is formally defined in section 2.1. Static clustering algorithms that take this approach include the depth first search (DFS) [Stamos 1984], breath first search (BFS) [Stamos 1984] and placement tree (PT) [Benzaken and Delobel 1990] algorithms. The object graph approach does not encapsulate information regarding navigations that do not follow the object graph and it also does not weight more frequently traveled paths of the object graph higher than the less frequently traveled. However, the advantage of this approach is reduced statistic overheads.

- *Statistical Object Graph (SOG):* In this approach, the object graph is weighed. The nodes are weighed according to frequency of object access and edges are weighted based on frequency of edge traversal. Two static clustering algorithms that use SOG are the weighted depth first search (WDFS) [Stamos 1984] and the cactus clustering algorithm [Drew et al. 1990]. The drawback of using SOG is its inability to model object co-references that do not follow the object graph.

- *Simple Markov Chain Model (SMC):* In this approach, the directed graph form of a first order Markov model is used as the clustering graph. Each accessible object in the system is represented by a node in the graph. Any positive probability that one object can be accessed after some other object, is represented as a directed edge. The node weights are labeled by probabilities of accessing the object and the edge weights are labeled by the transition probabilities ($\hat{P}(x,y)$ of section 3.5.2). The algorithms that use this approach include: the probability ranking principle algorithm (PRP) [Tsangaris 1992] (which only uses the node weights of the SMC model, see section 4.5.2); the Wisconsin greedy graph partition algorithm (WGGP) [Tsangaris 1992]; the Kernighan-Lin graph partitioning algorithm (KL) [Tsangaris 1992]; and the greedy graph partitioning algorithm [Gerlhof et al. 1993](see section 4.5.4).

5.2.2 Existing Buffer Replacement Algorithms

There have been numerous buffer replacement algorithms presented in the literature [Nicola et al. 1992; O'Neil et al. 1993; Belady 1966; Johnson and Shasha 1994; Robinson and Devarakonda 1990; Lee et al. 1999]. They mainly differ in the way they collect access information. The Least Recently Used (LRU) based policies measure the length of time between successive references of a page. The elapsed time between successive references of a page give a indication of when the page will next be referenced. A page with larger re-reference intervals is less likely to be needed in the near future, and thus would make a better candidate for eviction. LRU-K extends the basic LRU by recording historical information. The times of the K previous references are recorded. In addition, LRU-K removes the effects of correlated references[1]. LRU-K's historical reference information and removal of correlated references gives it superior performance compared to the basic LRU.

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

There are other simple algorithms that are designed to collect and store a small amount of information for performance reasons. These include, the First In First Out (FIFO) and CLOCK algorithms. The CLOCK algorithm is a popular replacement algorithm because of its simplicity and its ability to approximate the performance of the Least Recently Used (LRU) replacement policy. The CLOCK algorithm has been extended by GCLOCK [Nicola et al. 1992]. GCLOCK uses a circular buffer and a weight associated with each page brought into buffer to decide which page to replace. The weights may be different for different data types. The performance of GCLOCK can be better than LRU and performances very close to optimal can be achieved for some situations.

Finally there is Belady's optimal buffer replacement algorithm [Belady 1966]. This algorithm makes replacement decisions based on analysis of the entire reference trace. Therefore this algorithm can not be applied to real OODBMSs, however it is a useful tool of comparison.

[1] Correlated references occur when the same page is re-referenced in quick succession.

5.3 Preliminaries

In this section we first provide a formal definition of the problem we are attempting to solve. Second, we outline the assumptions the work in this chapter makes. Finally, the working set size metric is described in detail.

5.3.1 Problem Definition

The threads that we have are:

- This client thread (TC)
- Other client threads (OC)

Given a trace t_i, a buffer replacement algorithm and an interleaving $x_i(T_n)$, we seek to find the *static clustering algorithm* that minimises the execution time $ET(x_i(T_n),t_i)$ of t_i under $x_i(T_n)$ using equation 3.5 of section 3.4. This is formulated as:

$$min(ET(x_i(T_n),t_i)) = min(\sum_{r=0}^{h}(IO_{TCR}(r) + IO_{OCR})) \tag{5.1}$$

The write IO term $IO_{BW}(r)$ has been removed since we would like to concentrate on reducing client read IO. Static clustering algorithms are executed offline when the database is not in operation. Thus the computational and IO overheads of static clustering algorithms do not effect execution time of client threads. Hence clustering, CPU and IO overheads are not included in equation 5.1. However, it is still important for static clustering algorithms to possess low time complexity since there are normally limits on how long a database can be taken offline.

5.3.2 Assumptions

The work in this chapter makes the following assumptions:

1. The entire database can be shut off for rearrangement to take place.
2. Patterns of object access between rearrangements bear some degree of similarity.
3. All objects are smaller than one page in size. See assumptions in section 4.3.3.
4. Objects are moved and mapped from one consistent state to another.

5.3.3 Working Set Size Metric

We briefly described the working set size (WSS) metric in section 4.5.1. In this section we provide a more detailed description of the metric.

WSS was first used within the static clustering context by Tsangaris [1992]. It is a standard metric used to model locality [Denning 1968; Yue and Wong 1973]. A formal definition of WSS follows. $WSS(w)$ is the expected cardinality of the set $R_t^{(w)}$

of w consecutive page requests starting at time t. That is: take these w page requests, eliminate duplicates, and compute the cardinality of the resulting set. Note that the larger the cardinality, the fewer the duplicates, hence the lower the locality. $WSS(w)$ can be described mathematically as:

$$WSS(w) = E(\| R_t^{(w)} \|) \tag{5.2}$$

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

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

5.4 Cache Conscious Clustering Framework (C3)

In this section we describe the C3 framework. Firstly, the framework objective is defined and contrasted with the objective of existing comparable work. Secondly, the framework is defined.

5.4.1 Framework Objective

Customising static clustering for every possible buffer replacement technique is beyond the scope of this thesis. However, buffer replacement algorithms in general aim to retain in memory pages that are likely to be needed in the near future. Thus a static clustering algorithm that improves *locality* of the client reference stream should perform well for most buffer replacement algorithms. The WSS metric defined in section 5.3.3 is a metric that measures locality. We thus aim to find static clustering algorithms that minimise WSS.

Tsangaris [1992] first formulated static clustering as a minimisation of the WSS metric. However, they by-passed the synergy between static clustering and buffer replacement via two decisions. Firstly they only optimise for $WSS(2)$. This means given two consecutive page requests, the aim of clustering is to confine both requests to the same page. This formulation allows clustering to be optimised for cache sizes of *one* page. This is a boundary case and one in which buffer replacement is irrelevant. However, this approach still managed to inspire the graph partitioning family of clustering algorithms which offer best performance among all existing algorithms (see section 4.5.4). Second, Tsangaris [1992] did not test the effect of different buffer replacement algorithms. In fact, only one experiment uses a metric that is effected by buffer replacement algorithms.

The fundamental difference between our approach and that of Tsangaris [1992] is that we cluster for $WSS(w)$, where $w \geq 1$. Thus we cluster for caches larger than one page, where buffer replacement behaviour is relevant. We term static clustering algorithms that solve the $WSS(w), w \geq 1$ problem *cache conscious* clustering algorithms.

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

5.4.2 Framework Definition

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

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

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

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

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

1. *Define metric for regularly referenced objects.* This step provides a means of ranking objects based on how regularly they will be referenced. The best definition of this metric is likely to depend on the characteristics of the trace. For example a trace that contains approximately uniform object access intervals can use a simple metric like heat (the number of times the object has been referenced) as its regularity metric. A trace with non-uniform access behaviour may use the average time between references (smaller average time indicates higher regularity).

2. *Separate into N regions of homogeneous regularity.* This step divides objects into regions of similar regularity of reference. This is accomplished by sorting the objects into decreasing regularity of reference. The sorted sequence is then cut at N - 1 places to form N regions of more homogeneous regularity. The places at which the N - 1 cuts occur is a property of the particular C3 algorithm. See section 5.5 for an example.

3. *Cluster each region based on relatedness separately.* This step further divides each regularity region into pages using any clustering algorithm that clusters based on some notion of relatedness between objects. Most static clustering algorithms mentioned in section 5.2 fall into this category.

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

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

Figure 5.1 gives a diagrammatic representation of one example application of the C3 clustering framework. In the example, heat is used as the definition of regularity. The second and third steps of the C3 clustering framework are shown on the diagram. The second step divides the object base into two regions of contrasting heat. The third step clusters related objects further into pages.

5.5 Cache Conscious Greedy Graph Partitioning

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

1. *Define metric for regularly referenced objects.* C3-GP uses heat as the regularity metric. It assumes a uniform random reference distribution. That is, it assumes objects referenced with high regularity will have a higher total number of references than objects which are not accessed regularly. It is clear this assumption does not always hold in real life, since sometimes an object may not be referenced regularly but when it does get referenced it is hit many times. The search for the best clustering algorithm for every case is beyond the scope of this thesis. This thesis aims to demonstrate simple synergistic modifications to existing algorithm can result in improved performance. Extending the work in this chapter to more general cases is an interesting area of future work.

2. *Separate into N regions of homogeneous regularity.* C3-GP divides the object base into two regions of contrasting regularity. It labels the region with high regular-

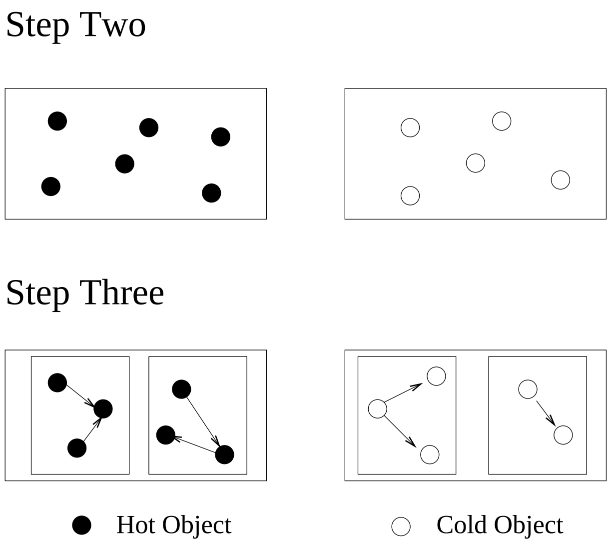


Figure 5.1: Diagrammatic description of the second and third steps of the C3 framework. In the example the heat is used as the regularity metric. The second step of this example only divides the objects into two regions of contrasting heat. However in general the second step of the C3 framework allows the object base to be divided into N regions.

ity the 'hot' region, and the other region the 'cold' region. The cut occurs at the point that results in all objects in the hot region just fitting in memory.

3. *Cluster each region based on relatedness separately.* C3-GP uses the graph partitioning algorithm GGP [Gerlhof et al. 1993] to further divide the objects in each region into pages. This step is particularly effective for reducing IO of the cold region. Graph partitioning the cold region increases the locality of reference of cold pages. Since cold pages are not expected to stay in memory long, it is appropriate to formulate the clustering of cold objects as a $WSS(2)$ minimisation problem. This effectively means objects belonging to the cold region are clustered for a cache of size one. Graph partitioning algorithms are the best algorithms for solving the $WSS(2)$ minimisation problem [Tsangaris 1992; Gerlhof et al. 1993].

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

5.6 Experimental Setup

The experimental setup used in this chapter is mainly the same as that reported in section 4.7. The main difference is that only the stationary access pattern evaluation environment is used (since we are only evaluating static clustering algorithms in this chapter).

The VOODB simulator and OCB benchmark are used to evalute the static clustering algorithms of this chapter. VOODB and OCB are described in sections 4.7.1 and 4.7.2.1 respectively. The VOODB simulator and OCB benchmark parameters used in this experiment are the same as those reported on tables 4.1 and 4.2 respectively.

In this chapter we compare the performance of the C3 algorithm C3-GP with three existing static clustering algorithms. The three existing static clustering algorithms are the probability ranking principle algorithm (PRP) [Tsangaris 1992], greedy graph partitioning (GGP) [Gerlhof et al. 1993], and Wisconsin greedy graph partitioning WGGP [Tsangaris 1992]. The reason for choosing these algorithms is they all use the SMC clustering graph. The SMC clustering graph algorithms has been shown by Tsangaris [1992] to give best general performance. The PRP and GGP algorithms are explained in sections 4.5.2 and 4.5.4 respectively. WGGP, like GGP is also a greedy graph partitioning algorithm, however it creates partitions in a different way. WGGP first sorts all objects in heat order. The algorithm starts by placing the hottest object into the first partition and then incrementally places the remaining objects as follows. Among all the objects that will fit into the current partition (partition size must be smaller than a page), find the object that has not been placed and has the highest edge weight[2] with the current partition. Place the selected object into the current partition. Repeat until no candidate objects can be found. Start a new partition using the hottest yet to be placed object as the first object and repeat the entire process.

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

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

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

The results were generated via three steps. The first *training* step runs the database and collects statistical data of object access. The second *clustering* step uses the train-

[2] Edge weight of the clustering graph, using SMC to model object level transitions.

ing data with the clustering algorithm to rearrange objects. The third *evaluation* step measures IO generated from running the workload on the newly clustered database.

The evaluation metric used is total IO. In this experiment total IO equals the total transaction read IO. Recall in section 5.3.1, that we are only interested in improving transaction read performance. The reason for using total IO instead of WSS as our evaluation metric is that ultimately we are interested in how well the algorithms can reduce total IO (see section 5.3.1). In this thesis the WSS metric is only used as a guide to explain the intuitions that led to the design of our algorithms.

5.7 Experimental Results

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

5.7.1 Varying Buffer Size

In this section we report the effects of varying buffer size on the performance of the static clustering algorithms. The buffer replacement algorithm used is the LRU algorithm. The results are shown on figure 5.2. The general observation is that C3-GP always performs better than or as well as the existing algorithms. C3-GP outperforms all existing algorithms between buffer sizes of 0.5MB and 5.8MB and shows equal performance for other buffer size settings. The reason for C3-GP performing the same as GP when the buffer size is less than 0.5MB is that at this smaller buffer size the buffer replacement algorithms find it difficult to retain pages belonging to the hot region of C3-GP in memory. This failure means clustering hot objects together is less profitable since even when many hot objects are clustered into the same page, the page still has a high probability of being evicted due to the small buffer size. When the buffer size is larger than 5.8MB almost all of the active portion of the database fits in memory and thus all static clustering algorithms perform about the same.

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

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

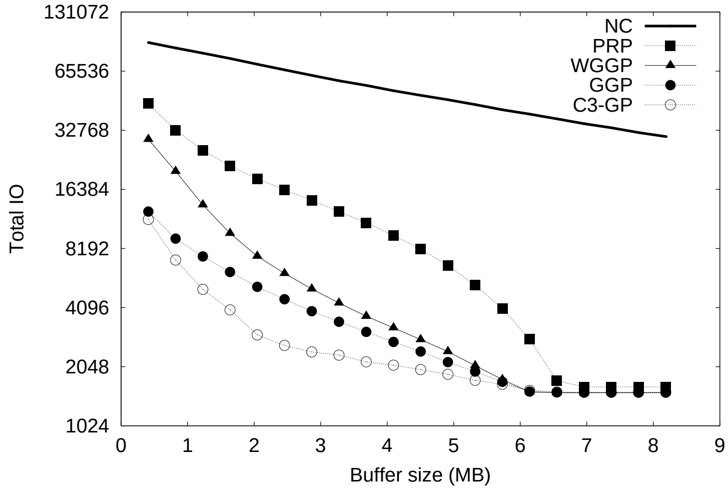
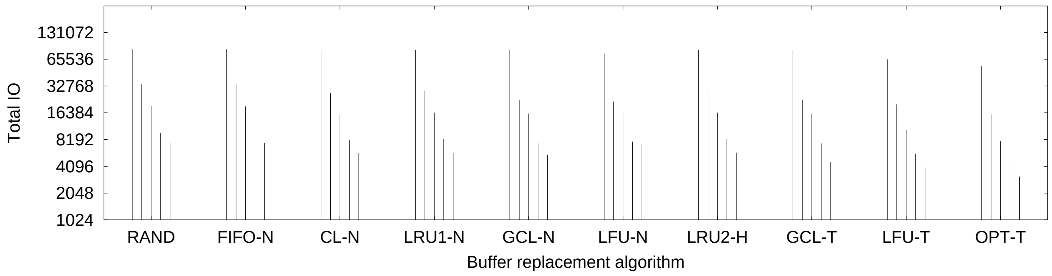


Figure 5.2: Varying buffer size experiment. The y-axis is in log_2 scale.

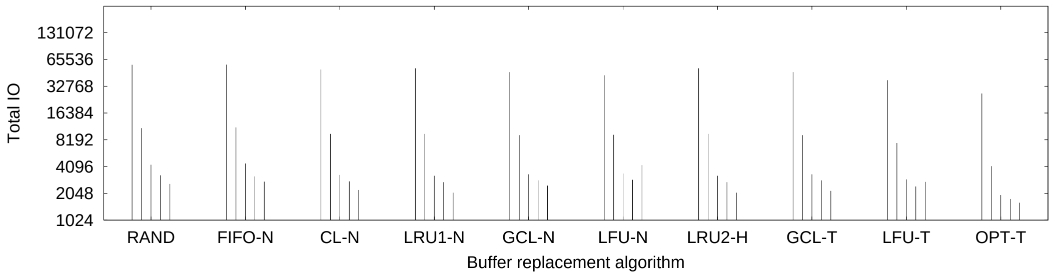
5.7.2 Varying Buffer Replacement Algorithm

In this section we explore the performance of the clustering policies: no clustering, PRP, WGGP, GGP and C3-GP on ten different buffer replacement algorithms. The ten different buffer replacement algorithms investigated include: random (RAND); First In First Out (FIFO-N); CLOCK (CL-N); traditional Least Recently Used (LRU1-N); GCLOCK (GCL-N) [Nicola et al. 1992]; Least Frequently Used (LFU-N); Least Recently Used K algorithm with K set to 2 (LRU2-H) [O'Neil et al. 1993]; GCLOCK algorithm using training data (GCL-T); Least Frequently Used algorithm using training data (LFU-T); Belady's optimum algorithm (OPT-T) [Belady 1966]. Algorithms with a T suffix use information gathered in the training step of the experiment to help make more accurate replacement decisions during the evaluation step. The N suffix is used for algorithms that do not use training data and also reset statistics for a page when it is first loaded into memory. Algorithms with a H suffix retains history information for a page when it is evicted from memory. However, H suffix algorithms do not use training data.

The results of using 1Mb and 4MB buffer sizes are reported on figure 5.3 (a) and (b) respectively. The results show that for the 1MB buffer size case, C3-GP offers best performance for all buffer replacement algorithms used. When the buffer size is 4MB, C3-GP is the best performer for 8 of the 10 buffer replacement algorithms used. The only cases in which C3-GP is not the best performer is when the buffer size is 4MB and the LFU-N and LFU-T buffer replacement algorithms are used. This is because at 4MB buffer size, almost all of the pages containing hot objects fit in memory, even when the hot objects are spread across many pages (the case with the NC, PRP, WGGP and GGP clustering algorithms). LFU algorithms which keep frequently accessed pages in memory prevents the pages containing hot objects from thrashing. Thus at these



(a) 1MB buffer size.



(b) 4MB buffer size.

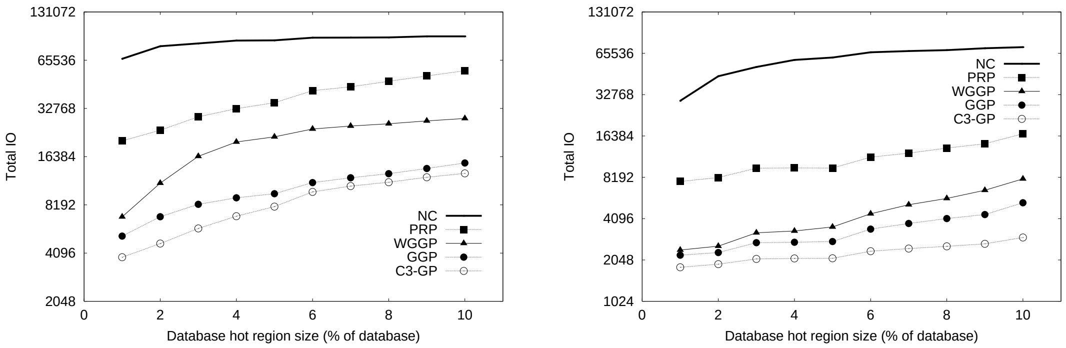
Figure 5.3: Varying buffer replacement algorithm experiment. The y-axis is in log_2 scale. The results for five different static clustering policies are reported for each buffer replacement algorithm result. The static clustering results are reported in the following order, no clustering, PRP, WGGP, GGP and C3-GP.

settings, C3-GP's ability to prevent thrashing of pages containing hot objects no longer gives it an advantage over the other algorithms. The result is that GGP and WGGP, which cluster solely based on relatedness, are able to meet the second sub-objective of section 5.4.2 better than C3-GP but do not suffer the negative consequences of not meeting the first sub-objective.

5.7.3 Varying Database Hot Region Size

In this experiment we varied the hot region size of the database and kept the probability of hot region access at a constant 0.8 (see section 4.7.2.1 for the reason behind choosing 0.8). The buffer replacement algorithm used is the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown on figure 5.4 (a) and (b). It is encouraging to observe C3-GP offers best performance for both 1MB and 4MB buffer sizes.

When using a buffer size of 1MB, C3-GP's performance lead over GGP diminishes



(a) 1MB buffer size.

(b) 4MB buffer size.

Figure 5.4: Varying database hot region size experiment. The y-axis is in log_2 scale.

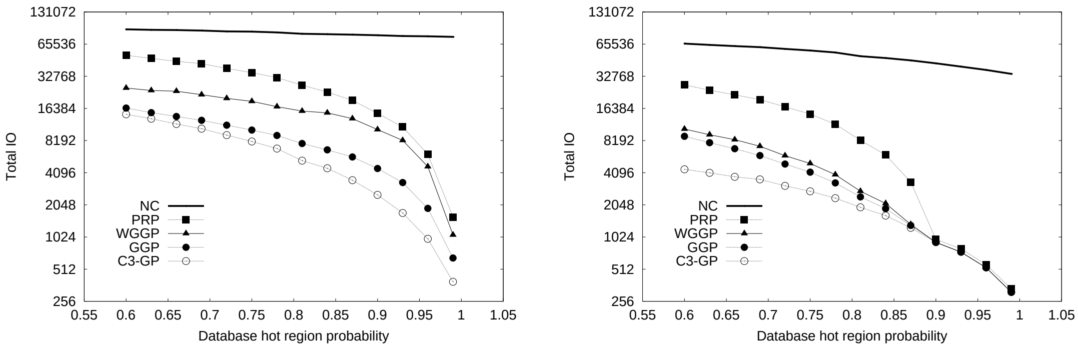
as the database hot region size increases. This is because as the hot region size increases, it becomes increasingly difficult for C3-GP to fit hot objects into *its* hot region. Thus many hot objects end up in cold pages. The result is that C3-GP is no longer able to prevent the thrashing of many of the pages that contain hot objects.

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

5.7.4 Varying Database Hot Region Access Probability

In this experiment we vary the probability of accessing objects in the hot region of the database. The size of the hot region is kept constant at 3% the size of the database (see section 4.7.2.1 for the reason behind choosing 3%). The buffer replacement algorithm used is again the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown in figure 5.5 (a) and (b). The results show that C3-GP offers the best performance in general.

At the 1MB buffer size, C3-GP exhibits the best performance for all the results reported. However, at the 4MB buffer size, C3-GP starts off well in front of the other algorithms but its lead diminishes as the database hot region probability increases. Eventually, at 0.9 all clustering algorithms perform the same. This is because at above 0.9 hot region access probability, almost all queries are confined to the hot region. Since the hot region is relatively small compared to the 4MB buffer size, the entire



(a) 1MB buffer size.

(b) 4MB buffer size.

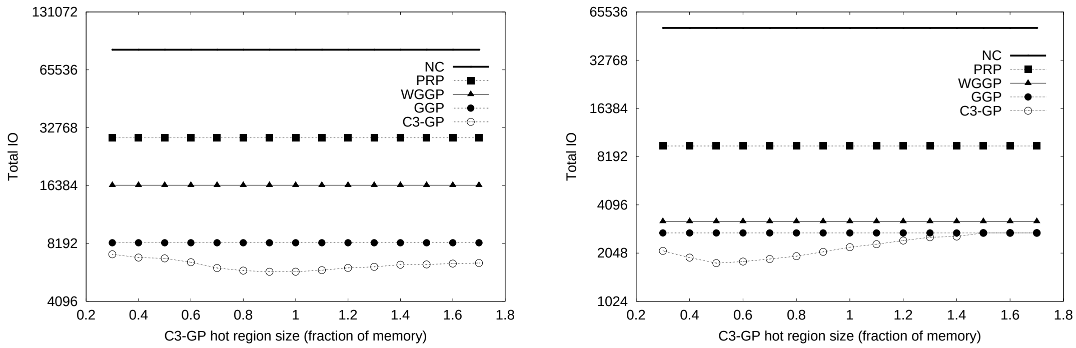
Figure 5.5: Varying database hot region access probability. The y-axis is in log_2 scale.

active portion of the database fits in memory. All of the static clustering algorithms are able to group the active portion of the database together and away from the non-active portion. This explains why all of the algorithms perform the same when the database hot region access probability is above 0.9.

5.7.5 Varying C3-GP Hot Region Size

In this experiment we varied the size of C3-GP's hot region. Recall from section 5.5, C3-GP's hot region is created by sorting the objects in decreasing heat and then taking the top x objects as belonging to the hot region. In the definition of C3-GP, x is chosen so that all of the objects just fit into memory. In this experiment we vary the place where the sorted list of objects is cut. The buffer replacement algorithm used is the LRU algorithm. The results when using the 1MB and 4MB buffer sizes are shown in figure 5.6 (a) and (b). The results for NC, PRP, WGGP and GGP do not change when C3-GP hot region size is varied, since these algorithms do not use this parameter.

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



(a) 1MB buffer size.

(b) 4MB buffer size.

Figure 5.6: Varying C3-GP hot region size. The y-axis is in log_2 scale.

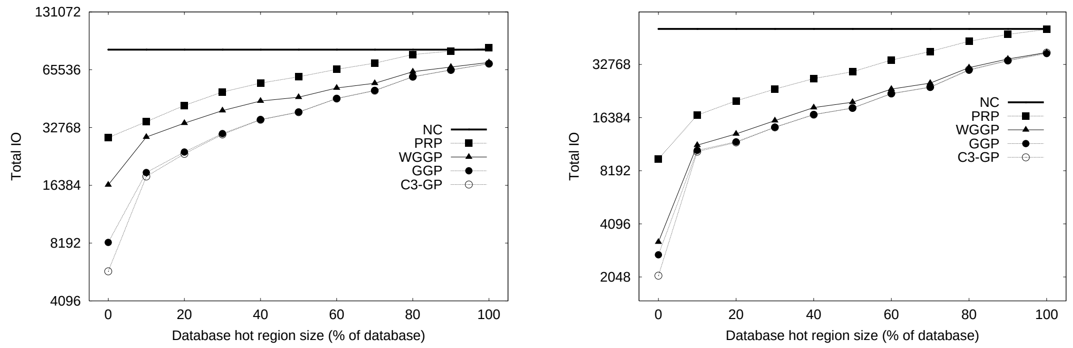
5.7.6 Training Skew

Until now all of the experiments involved running the same set of transactions for both the training and evaluation steps. In contrast, this experiment explores the effect of running a different set of transactions for the training and evaluation steps. This is achieved by moving the hot region of the database. The numbers on the x-axis of figures 5.7 (a) and (b) show by how much the database hot region is moved. For example, a value of 20% means 20% of the hot region used for the training step became part of the cold region used for the evaluation step. This gives an indication of the degree of difference between transactions used in the training and evaluation steps. The hot region size of database was set to 3% of database size. The buffer replacement algorithm used is the LRU algorithm.

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

5.7.7 Discussion

The results show that C3-GP outperforms existing static clustering algorithms in a variety of situations. Among the situations tested are 10 different buffer replacement algorithms, various buffer sizes, database hot region sizes, access probabilities, C3-GP's hot region sizes and various amounts of training skew. Among all of the experimental results, C3-GP performed at least as good as the existing algorithms for all but one



(a) 1MB buffer size.

(b) 4MB buffer size.

Figure 5.7: Varying training skew. The y-axis is in log_2 scale.

particular situation (when the LFU-N and LFU-T buffer replacement algorithms are used and the buffer size is large). This ability to perform well so consistently makes C3-GP ideal for deployment in general purpose OODBMS in which workload conditions and system settings are not known apriori.

5.8 Conclusion

In this chapter we highlight the performance gains resulting from exploiting the synergies between *static clustering* and *buffer replacement*. To this end we describe the cache conscious clustering framework (C3). C3 produces static clustering algorithms which use knowledge of how buffer replacement algorithms behave to make clustering decisions. Like OPCF of the previous chapter, C3 is simple and straightforward to apply and general in that it can be used to produce a whole family of different static clustering algorithms. Using the C3 framework, we produced a new static clustering algorithm (C3-GP) which outperforms highly competitive existing algorithms in a variety of situations.

The next chapter investigates the advantages of exploiting the synergies between *prefetching* and *buffer replacement*.

Chapter 6

Path and Cache Conscious Prefetching Framework (PCCP)

The previous two chapters explored the advantages of exploiting the synergies between static and dynamic clustering; and static clustering and buffer replacement, respectively. The approach taken in both of those chapters was to develop simple and general frameworks by which a family of new algorithms can be created.

This chapter uses the same approach as the previous two chapters, namely a simple and general framework for exploiting synergy. However, this time the synergy between *prefetching* and *buffer replacement* is explored. This is done by creating a general framework that produces prefetching algorithms which are cache conscious — aware of which types of pages are likely to be kept in memory by the buffer replacement algorithm. In addition to cache consciousness, the algorithms produced by this framework are also path conscious — they incorporate cheap path information to allow prefetches to be started earlier and with more accuracy. Accordingly, we term our new prefetching framework, 'path and cache conscious prefetching' (PCCP).

This chapter first uses the integrated cost model of section 3.4 to define the problem statement. Second, we propose a new metric called the prefetch quality metric (PQM) to aid in explaining the intuition behind the prefetch algorithms. Third, the metric is used to explain how the concepts used by PCCP can reduce disk IO stall time. Fourth, the PCCP framework is defined. Fifth, four new prefetching algorithms produced from the PCCP framework are presented. Finally, a simulation study comparing PCCP algorithms with two existing highly competitive prefetching algorithms is presented.

6.1 Introduction

Prefetching is a promising technique for tackling the IO performance bottleneck. It works by predicting the next non-memory resident page request in advance and then pre-loading that page in the background. This approach allows disk IO to be overlapped with CPU, thus reducing disk IO stall time.

Central to the design of prefetching algorithms is the design of the prediction engine. Most existing prefetching algorithms for ODBMSs use a training-based prediction engine [Curewitz et al. 1993; Knafla 1998; Palmer and Zdonik 1991; Han et al. 2001]. Training-based prediction engines use historical access information to make future prefetching decisions. There are two problems with existing training-based prediction engines:

- The high storage cost of storing fine grained (object[1] grained) access statistics.
- The small time gap between prediction and reference of the next prefetched page.

Storing access statistics at the fine object grain [Bernstein et al. 1999; Chang and Katz 1989; Knafla 1997; Knafla 1998; Palmer and Zdonik 1991] can provide prediction engines with more precise statistical information of access patterns. However, storing access information at the object grain incurs high storage overheads.

Another problem with existing prediction engines is the small time gap between prediction and reference of the next prefetched page. The consequence is that there is small potential overlap between IO and CPU. Existing prediction engines can only predict the next disk page request a few object references ahead of time. This limitation is becoming a bigger problem, since the rate of CPU improvement is much greater than that of disk IO. Therefore the CPU time it takes to process each object becomes much smaller relative to one disk IO[2]. This in turn leads to a smaller amount of overlap between CPU and disk IO for a prefetch started the same number of object references in advance. The path and cache conscious prefetching framework (PCCP) addresses both of these deficiencies in existing prefetching algorithms.

During prediction engine training, PCCP minimises statistics storage by storing short sequences of object references at key points (which we term 'feature points') in the reference trace. When these feature points are later encountered at prefetch time, the stored statistics are used to decide if a prefetch should be started. These feature points are selected to be sparsely spaced and early in terms of when the next prefetched page will be referenced.

There are two key concepts in PCCP, *path* and *cache* consciousness. These concepts are both introduced for this first time in this thesis. Path consciousness refers to the careful selection of feature points so that the current path of navigation can be identified early and cheaply. In cache conscious prefetching, historical page access knowledge is used to guess which pages are likely to be cache resident most of the time (we term these pages 'resident' pages) and these pages are then ignored in the context of prefetching. Therefore cache conscious prefetching reduces the number of feature points to only those that occur during traversal of pages deemed to be 'non-resident', thereby reducing the total volume of statistics stored. An even more important result of cache consciousness is that only 'non-resident' pages are candidates for prefetching. The implication of this property is that prefetches can be started earlier (section 6.4.2 explains the reason for this behaviour).

[1] From this point on, we will use the term 'object' to refer to the more generic term 'data item'.
[2] Assuming the amount of computation per object remains the same.

6.2 Related Work

Gerlhof and Kemper [1994] identify two dimensions along which prefetching algorithms can be classified: prediction engine used; and granularity of prediction.

Prediction engines are typically divided into three types: *strategy-based*; *structure-based*; and *training-based*.

Strategy-based prefetching algorithms use explicitly programmed strategies to decide which objects to prefetch. The simplest example is the one block lookahead algorithm (OBL) [Joseph 1970], which upon a demand fetch[3], prefetches the next adjacent block. Another example is Thor's prefetching policy [Liskov et al. 1996]. Thor divides objects into prefetch groups and whenever an object in a prefetch group is requested, all the objects in the group are fetched. More recently, Bernstein, Pal, and Shutt [1999] proposed a new strategy-based prefetching algorithm termed context-based prefetching. Context-based prefetching fetches all objects in the structure context of the requested object. Examples of structure context include query results and collections. The general problem with strategy based prediction engines is their lack of flexibility in catering for different paths of object graph navigation.

Structure-based prefetching algorithms obtain information from object structure. In the Chang and Katz [1989] approach, objects are linked via three types of structural relationships: inheritance; configuration; and version history. The user specifies which type of relationship he/she is currently navigating under and this information is used in combination with the structural hierarchy graph to decide which objects to prefetch next. The problem with this approach is its reliance on user provided information and its eagerness in prefetching (all component objects that are decedents of the current object under the current user defined structural relationship type are prefetched). Knafla [1997] proposes an approach in which different possible paths of navigation are first identified *using the object graph alone* and then as client navigation proceeds, the prefetch algorithm uses the current navigational context to determine which path is likely to be taken. All structure-based approaches assume objects will always be accessed via navigating references defined in the object graph, however, many ad hoc queries do not navigate the object graph. Thus structure-based approaches perform poorly when queries do not navigate the object graph.

Training-based prefetching algorithms uses historical access information to make future prefetching decisions. Training-based prefetching algorithms can be further divided into three different categories: *object-sequence-based*; *compression-based*; and *type-level-based*.

Object-sequence-based prefetching algorithms use observed object access patterns to make future object access predictions. The Fido algorithm [Palmer and Zdonik 1991] prefetches by identifying and matching sequences of object accesses and storing them as patterns in pattern memory. However, their pattern memory mechanism of storing access sequences is expensive. The PMC prefetching algorithm [Knafla 1998] uses discrete-time Markov chains (DTMC) to model object-level access patterns. Since

[3] When a page is loaded upon request and no sooner.

DTMC only allows the prediction of future accesses based on the current state, they only incorporate path information of length one. This approach is expensive in terms of statistics storage cost (using DTMC to model *object-level* transitions) and makes predictions late (short time between prediction and when the next prefetched page is referenced).

Compression-based prefetching algorithms [Curewitz et al. 1993] use the principles of data compression for training and prediction. The intuition is that data compressors typically operate by predicting the dynamic probability distribution of the data to be compressed. If the data compressor successfully compresses the data, then its predicted probability distribution must be correct and can then be used for prediction in prefetching. One such algorithm is the predicition-by-partial-match (PPM) prefetching algorithm [Curewitz et al. 1993]. PPM uses a predictor-based on the higher order Markov chains (HMC) model (see section 3.5.3). Curewitz, Krishnan, and Vitter [1993] found that PPM gave the best performance among the compression-based algorithms. In addition, PPM was found to outperform Fido. The problem with PPM is the coarse grain (page grain) at which statistics are stored. This coarse granularity of prediction produces less accurate prediction engines when compared to PCCP algorithms (which uses a hybrid object/page grained predictor).

Han, Whang, Moon, and Song [2001] proposed a *type-level-based* prefetching algorithm for ORDBMSs. In their algorithm, recurring access patterns at the type-level are first identified and then used for prediction. Type-level access patterns are patterns of attributes that are referenced when accessing the objects. The drawback of this approach is its dependency on type-level access locality. Many ODBMS applications issue short ad hoc queries to the database, these queries do not exhibit type-level access locality.

The second dimension of prefetching algorithm classification is granularity of prediction. There are three typical grains of prediction: object grain; page grain; and attribute grain. Object grained techniques [Bernstein et al. 1999; Chang and Katz 1989; Knafla 1997; Knafla 1998; Palmer and Zdonik 1991] make predictions using object level information. Page grained algorithms [Curewitz et al. 1993; Joseph 1970] observe access patterns that occur at the page level (popular in file systems research). Attribute grained algorithms use patterns of attributes to make predictions, eg. the type-level-based prefetching algorithm in Han, Whang, Moon, and Song [2001].

Cao, Felten, Karlin, and Li [1995] study the implications of integrating *prefetching* and *buffer replacement* when perfect knowledge of future access sequence is known. However, their research is not applicable to our work since we do not assume perfect knowledge of future access patterns. See section 1.3 for a more detailed explanation.

6.3 Preliminaries

In this section we first provide a formal definition of the problem we are attempting to solve. Secondly, we outline the assumptions that the work in this chapter makes. Finally, we define a new metric called the prefetch quality metric (PQM).

6.3.1 Problem Definition

Using the integrated cost model of section 3.4 we now formally define the problem.
The threads that we have are:

- This client thread (TC)
- Other client threads (OC)
- Prefetcher thread (P)

Given a trace t_i, an initial object to page mapping (intial clustering), a buffer replacement algorithm and an interleaving $x_i(T_n)$, we seek to find the *prefetching algorithm* that minimises the execution time $ET(x_i(T_n), t_i)$ of t_i under $x_i(T_n)$ using equation 3.5 (page 18). This is formulated as:

$$\mathrm{Min}(ET(x_i(T_n), t_i)) = \mathrm{Min}(\sum_{r=0}^{h} IO_{TCR}(r) + \sum_{r=0}^{h}(IO_{PIR}(r) + CPU_P(r)) + \sum_{r=0}^{h} IO_{OCR}(r)) \quad (6.1)$$

The $CPU_{TC}(r)$ and $CPU_{OT}(r)$ terms from equation 3.5 have been omitted because in this chapter we are interested in finding the best prefetch algorithm. Prefetching has no effect on the amount of CPU time required by the client threads.

The write IO term has been removed from equation 3.5 since prefetching does not directly affect the amount of write IO required, although prefetching does affect write IO behaviour indirectly by changing the order in which pages are loaded into memory. However, these effects are minor and will be left out of this chapter for simplicity.

6.3.2 Assumptions

The work in this chapter makes the following assumptions:

1. There is no concurrency at the disk IO level. That is, only one disk IO can occur at a time.
2. The patterns of object access do not change at a fast rate.
3. Objects do not move from one page to another. Removing this assumption is an area for future work. It should not be difficult, just requiring techniques that re-adjust or reset prefetching statistics when an object is moved from one page to another.

6.3.3 Prefetch Quality Metric (PQM)

In equation 6.1, the $IO_{PIR}(r)$ term represents the time this client is blocked due to incorrect prefetch IO by the prefetcher thread between references r and $r+1$. Incorrect prefetch IO is defined as a prefetched page not corresponding to the next disk page request. However, this definition of incorrect prefetch IO does not incorporate the benefits of prefetching a page which is first referenced on the 2nd, 3rd, etc., disk load after the prefetch.

We now define a new metric that scales the prefetching benefit based on how soon it is referenced after being prefetched. The metric is called the prefetching quality metric (PQM). This metric may allow algorithm developers to gain a better understanding of the reasons behind prefetching algorithm performance. It may also be used as a starting point for algorithm developers when tackling the prefetching problem. PQM is used in this thesis as a tool for explaining the intuition behind the various prefetching algorithms. The PQM metric is defined below:

$$PQM = \sum_{i=0}^{NL-1} \left(v(i) \times \frac{f(memsize - NDL(i))}{memsize} \right) \tag{6.2}$$

$$f(x) = \begin{cases} x & \text{if } NDL(i) < memsize \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

NL is the number of page loads. $v(i)$ is the overlap between CPU and disk IO during the i^{th} page load[4], specified in time units. *memsize* is the size of main memory in terms of number of pages. $NDL(i)$ is the time (in terms of number of disk page loads) between the i^{th} page load and when it is first referenced.

The intuition behind *PQM* is that a page prefetched but not used for a long time (in terms of number of disk page loads) wastes main memory space which may cause premature eviction of in memory pages. Thus the quality of prefetching is scaled based on how long after loading a prefetched page it is first referenced. If the prefetched page is the first non-memory resident page referenced after or during the prefetch, then the scale factor (second part of equation 6.2) is assigned a maximum value of one. The scale factor is assigned a value of zero when the prefetched page is not referenced until after *memsize* number of pages have been loaded from disk.

6.4 Path and Cache Conscious Prefetching Framework (PCCP)

In this section we first describe the concept of path and cache conscious prefetching. We then explain how PCCP increases prefetch quality. Finally, we define the PCCP framework.

6.4.1 The Concept

This section introduces the two key concepts, *path* and *cache conscious* prefetching. Both concepts rely on historical training data to gain insight into how the database is being used. The training data is then used during prefetching.

In path conscious prefetching, features in the object trace are remembered during training and used to identify the current path of navigation during prefetching. The

[4]The amount of time that the CPU spends processing objects while the i^{th} page is being loaded.

current navigational path information can then be used to determine the next non-memory resident page to be prefetched. The goal of path conscious prefetching is to identify the current path of navigation as early and accurately as possible.

Cache conscious prefetching uses training data to divide the pages of the database into two types: *'resident'*; and *'non-resident'*. *'Resident'* pages are deemed to be always memory resident (however, in practice they will sometimes be non-memory resident but this should occur rarely). In contrast, *'non-resident'* pages are deemed to be always non-memory resident (again in practice these pages are sometimes in memory). Having divided the database into 'resident' and 'non-resident' pages, cache conscious prefetching is only interested in prefetching the 'non-resident' pages. Since 'resident' pages are almost always in memory, avoiding them completely will cost only a small number of prefetch opportunities. The benefits of such an approach are that prefetching can be started earlier and prefetching overheads are lowered (only storing statistics for 'non-resident' pages). The reasons this approach allows prefetching to be started earlier are described in section 6.4.2.

Path and cache conscious prefetching can be used in a complementary fashion. First, cache conscious prefetching is used to classify pages as either 'resident' or 'non-resident'. Then path conscious prefetching gathers prefetching statistics using a limited scope ('non-resident' pages only). This approach provides the benefits of both path and cache conscious prefetching.

6.4.2 How PCCP Increases Prefetch Quality

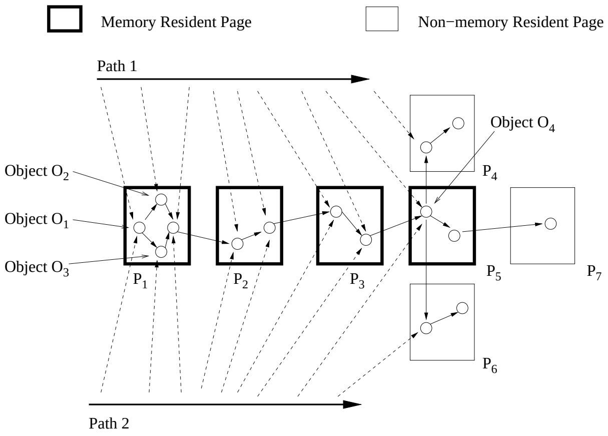
In this section we compare PCCP with two existing highly competitive training based prefetching algorithms, PPM-2 and PMC (see section 6.2). Given the same example object base navigation we show how PCCP produces higher prefetch quality metric (PQM) values than PPM-2 and PMC. Assume in this example that it takes 11ms[5] to load a page from disk into memory, and 1ms [6] to process one object. PQM calculations are done in time units instead of ms so for this section we will equate 1ms with 1 time unit. Also assume main memory can fit 5 pages.

The reader is reminded, as they read ahead, that PPM and PMC only allow prediction of prefetch pages *one page ahead*. This means the earliest the prefetch of page p_2 can be started is when page p_1 begins to be referenced, given that p_2 is referenced immediately after p_1. In contrast, and depending on the circumstances, cache conscious clustering allows prefetching to be started well before p_1 begins to be referenced. The example below will explain this behaviour more clearly.

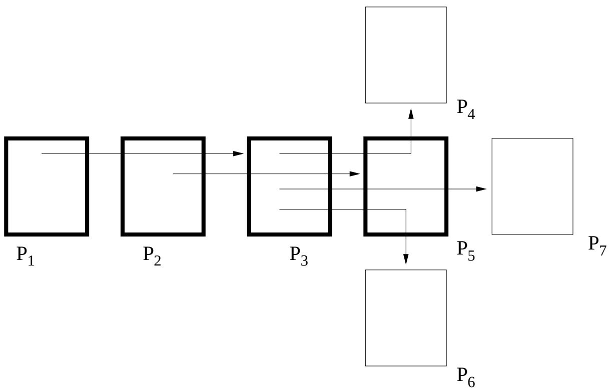
Figure 6.1 contrasts the statistics collected by the prefetching algorithms, PMC, PPM-2, and PCCP, given the same example object base navigations. Figure 6.1(a) con-

[5]Typical current hard disk performance which is also used for our experiments. The 11ms page load time is derived from the summation of 6.5ms, 4ms and 0.5ms, which are seek, latency and page transfer times respectively.

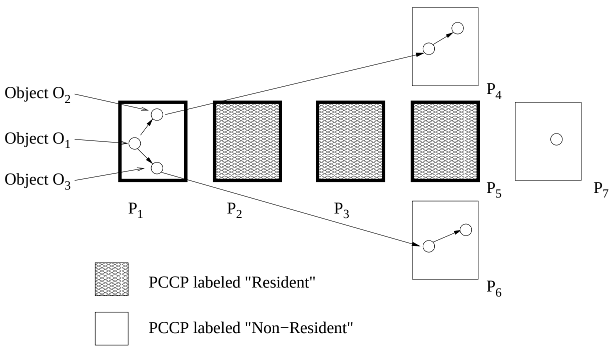
[6]This value varies based on application behaviour and system parameters such as CPU speed. In this example our PCCP algorithm outperforms PPM and PMC (in terms of the PQM metric) for any object processing time value greater than zero.



(a) PMC Prefetching



(b) PPM-2 Prefetching



(c) PCCP Prefetching

Figure 6.1: Illustration of the statistics collected for each prefetching algorithm, given the same example object base navigations.

veys both the illustration of two paths of navigations (path 1 and path 2) and PMC's object transition statistics. PMC uses an object-level discrete-time Markov chains (DTMC) model to make predictions. This means PMC only stores object transition statistics between consecutive pairs of object references.[7] These statistics are stored in a object transition graph in which nodes represent objects and weights on edges represent probability of traversal. In order to avoid cluttering the figure, transition probabilities are not depicted. However, for the purposes of this example it is sufficient to assume that all probabilities are some number greater than zero but less than one. Now assume navigations starting from object O_1 always either follow path 1 or path 2. It should be possible to prefetch either page P_4 or P_6 depending on which object is referenced after object O_1. Page P_4 should be prefetched if the access sequence is O_1, O_2. Similarly, the access sequence O_1, O_3 should predict page P_6. The problem with using PMC is that the statistics collected only capture the probability of transiting from one object to the next. This means that using PMC's statistics we cannot prefetch with total confidence until we observe which O_4 branch is taken (from object O_4 there are three different possible objects to reference next). Therefore, in this example PMC cannot perform any prefetching, resulting in a PQM value of zero (since $v(i)$ equals zero[8]).

The PPM-2 prefetching algorithm (shown in figure 6.1(b)) has the same problem as PMC. PPM-2 prefetching collects only page level transition statistics. At page P_5 there are three possible next pages, P_4, P_6 and P_7. Therefore, none of the pages can be prefetched with complete confidence. Like PMC, PPM-2 also produces a PQM value of zero.

Figure 6.1 (c) depicts the statistics that can be collected by PCCP algorithms. As a result of cache conscious prefetching, PCCP has deemed pages P_2, P_3 and P_5 as memory resident and therefore are ignored for prefetching purposes. Then combining this knowledge and path conscious prefetching (where features in the object trace are used to distinguish between different paths of navigation), the statistics shown on Figure 6.1 (c) can be collected. Using these statistics, we can start the prefetch of page P_4 as soon as O_2 is referenced, since the statistics collected capture the knowledge that the sequence O_1, O_2 predicts page P_4. Similarly, the prefetch of page P_6 can be started once O_3 is referenced. The following equation shows how we calculate PCCP's PQM value:

$$\begin{aligned} PQM &= \sum_{i=0}^{NL-1} \left(v(i) \times \frac{f(memsize - NDL(i))}{memsize} \right) \\ &= 7 \times \frac{f(5-0)}{5} + 7 \times \frac{f(5-0)}{5} \\ &= 14 \end{aligned} \tag{6.4}$$

[7]Prefetching algorithms generally do not store longer object level sequences, due to high storage overheads.

[8]The page being prefetched is needed straight after the prefetch starts, thus precluding any overlap between IO and CPU.

In equation 6.4, NL equals 2 since we are only considering the loading of pages P_4 and P_6. We substitute $v(0)$ and $v(1)$ with 7. The 7 unit overlap between CPU and disk IO is derived from the fact that 7 objects are processed between when the prefetch is first started (either at O_2 or O_3) and the loading of the non-memory resident page (P_4 or P_6 respectively). $NDL(0)$ and $NDL(1)$ both equal zero since in both cases the prefetched page is also the first non-memory resident page requested.

The ability for PCCP to produce a PQM value of 14 relies on the fact that PCCP has correctly labeled the pages of the database as either 'resident' or 'non-resident'. However, lets suppose P_3 was wrongly labeled, that is P_3 is actually non-memory resident but was labeled as memory 'resident'. In this case the following PQM value would be produced:

$$\begin{aligned} PQM &= \sum_{i=0}^{NL-1} \left(v(i) \times \frac{f(memsize - NDL(i))}{memsize} \right) \\ &= 4 \times \frac{f(5-1)}{5} + 4 \times \frac{f(5-1)}{5} \\ &= 6.4 \end{aligned} \tag{6.5}$$

In equation 6.5, $NDL(0)$ and $NDL(1)$ equals 1 since the prefetched page is the second non-memory resident page requested (see definition of $NDL(i)$ in section 6.3.3).

The above example shows that PCCP can outperform PMC and PPM-2 (which both give a PQM value of zero) even when it wrongly labels a page. This is due to the fact that PCCP has predicted correctly and early that the navigation will soon proceed to page P_4 or P_6 (depending on which navigational path was taken), despite the fact pages P_4 and P_6 (depending on navigational path) are not the next non-memory resident pages requested.

This example demonstrates how PCCP can start a prefetch much earlier than the state of the art prefetching algorithms PMC and PPM-2. In our simulation study (section 6.7) we found that situations similar to this example occur often. Frequently, many consecutive 'resident' pages references occur before a 'non-resident' page is referenced; and the first couple of object references in a page can uniquely identify the next disk page referenced.

6.4.3 Framework Definition

In this section we describe the PCCP framework. The PCCP framework allows the definition of a family of prefetching algorithms which all possess path and cache consciousness. PCCP prefetching algorithms use training-based prediction engines and store statistics at both the object and page grain. Object grained statistics are used for feature point selection. Page grained statistics are used to classify database pages as either memory 'resident' or 'non-resident'.

In order to define a PCCP prefetching algorithm, the following steps must be followed:

- ***Define feature point selection algorithm:*** In this step, an algorithm is defined for finding feature points in the trace. Feature points are object sequences occurring at special points in the trace. A feature point can span one or more pages. During prediction engine training, feature points are identified and stored, together with the page that the feature point predicts. For example, a feature point selection algorithm that picks the first two object references occurring in a page as a feature point stores the following statistics: at every page reference, the object ID of the first two objects referenced is stored (in sequential reference order) together with the page ID of the next page referenced. Feature point selection algorithms should aim to select feature points that can distinguish different paths of navigation as early and cheaply (both CPU and storage costs) as possible.

- ***Define 'resident' / 'non-resident' page metric:*** Cache conscious prefetching requires the classification of database pages as either memory 'resident' or 'non-resident'. In this step, a metric is used to rank pages in terms of likelihood of being memory resident at any moment in time. Metrics include: frequency of page references; sum of past memory residency durations; and hot/cold page classification information given by C3 clustering algorithms (see section 6.5 for a PCCP prefetching algorithm that uses this metric). Database pages are sorted according to this metric in descending order and the first *MEM_RES_PAGES* pages are classified as memory 'resident', the remaining pages are classified as memory 'non-resident'. A possible basis for choosing *MEM_RES_PAGES* is via physical memory size, e.g. *MEM_RES_PAGES* multiplied by page size should equal 90% of physical memory.

- ***Define prefetch threshold:*** If the probability of next navigating to a particular 'non-resident' page is greater than the prefetch threshold (*PREF_THRESHOLD*), that page is prefetched. The prefetch threshold is user defined.

At prefetch time the prediction engine looks for feature points occurring in 'non-resident' pages. When one is found, the corresponding training data is loaded and used to find the next 'non-resident' page with the largest probability of being referenced. If that page's probability of reference is greater than *PREF_THRESHOLD*, the page is prefetched.

6.5 Four New Concrete PCCP Algorithms

In this section we describe four new prefetching algorithms created from the PCCP framework. These four algorithms are derived from the following PCCP design decisions:

- ***Feature point selection algorithm:*** We define two alternative feature point selection algorithms. The term 'entry object' is used in this section to describe the first object referenced in each page.

Algorithm Name	Feature selection	Residency metric
Integrated path 1 prefetching (IP1)	FOR	CB
Integrated path 2 prefetching (IP2)	FORTP	CB
Heat based path 1 prefetching (HP1)	FOR	HB
Heat based path 2 prefetching (HP2)	FORTP	HB

Table 6.1: Four PCCP prefetching algorithms.

- *First object referenced (FOR):* In this simple policy, the entry object is selected as the feature point. During prediction engine training, the object ID of the entry object is stored together with the probabilities of navigating to each next 'non-resident' page. This simple policy performs surprisingly well in our experimental study (see section 6.7).
- *First object referenced in two pages (FORTP):* In this policy sequences of two consecutive entry objects are defined as a feature points. During prediction engine training, the object IDs of two consecutive entry objects are stored (in sequential reference order), together with the probabilities of navigating to the next 'non-resident' page.

• ***'Resident' / 'non-resident' page metric:*** We define two alternative 'resident' / 'non-resident' page metrics.

- *Heat based (HB):* In this approach we use page heat (where 'heat' is simply a measure of access frequency) as the 'resident' / 'non-resident' page metric. This is based on the observation that in general, frequently referenced pages are less likely to be evicted at buffer replacement time.
- *Clustering based (CB):* In this approach we use clustering information to determine whether a page is 'resident' or 'non-resident'. More specifically, clustering information from the C3-GP clustering algorithm is used (see section 5.5 a detailed description of the C3-GP algorithm). C3-GP first divides the database into hot and cold regions, then clusters objects of each region into pages separately. In this approach we classify all pages in C3-GP's hot region as 'resident' pages and the remaining pages as 'non-resident'.

Using the design alternatives above, we produced four new prefetching algorithms. Table 6.1 displays the design decisions each prefetching algorithm made.

6.6 Experimental Setup

In this chapter we mainly use the same experimental setup (with a few modifications) as those reported in section 4.7. The primary modification is made in the VOODB simulator [Darmont and Schneider 1999].

Parameter Description	**Value**
System class	Centralised
Disk page size	4096 bytes
Buffer size	varies
Buffer replacement strategy	LRU
Pre-fetching policy	varies
Object initial placement	Optimised sequential
Object think time	1 ms
Disk seek time	6.5 ms
Disk latency	4.3 ms
Disk transfer time	0.5 ms

Table 6.2: VOODBM parameters. In contrast to experiments in previous chapters, we include VOODB 'time' parameters. The reason for this is that in this chapter we are interested in the total IO stall time instead of just total IO.

VOODB's prefetching simulation framework is not fully developed. Consequently we have extended the simulator to allow full support for our prefetch algorithms. The extended simulator has been validated against example traces we have created and computed the IO stall time for. The VOODB parameters we have used for the experiments in this chapter are shown on table 6.2.

The results are generated via four steps. The first *clustering training* step runs the database and collects clustering statistical data. The second *clustering* step uses the training data with the clustering algorithm to rearrange objects. The third *prefetching training* step runs the newly clustered database to collect prefetching statistical data. The fourth *evaluation* step runs the prefetching algorithm with the newly clustered database to measure the performance of the system. In one of the experiments we have changed the trace used for the third and fourth steps to measure each prefetching algorithm's ability to adapt to changes in access pattern.

This chapter uses the OCB benchmark to compare the performance of the prefetching algorithms (see section 4.7.2.1). The OCB benchmark parameters we use are the same as those shown on table 4.2.

Most of the experiments in this chapter include two sets of results, one set uses the C3-GP clustering algorithm and the other set uses a combination of three clustering policies, greedy graph partitioning (GGP) [Gerlhof et al. 1993], Wisconsin greedy graph partitioning WGGP [Tsangaris 1992], and no clustering. The C3-GP, WGGP and GGP algorithms are explained in sections 5.5, 5.6 and 4.5.4 respectively.

The prefetching algorithms shown in the result graphs of this chapter are labeled as follows:

- **DM**: demand paging;
- **PPM-1**: first order prediction-by-partial-match prefetching [Curewitz et al. 1993];
- **PMC**: Knafla's [Knafla 1998] object grained statistical prefetching technique;
- **PPM-2**: second order prediction-by-partial-match prefetching [Curewitz et al. 1993];
- **PCCP-IP1**: IP1 prefetching algorithm, see section 6.5;
- **PCCP-IP2**: IP2 prefetching algorithm, see section 6.5;
- **PCCP-HP1**: HP1 prefetching algorithm, see section 6.5;
- **PCCP-HP2**: HP2 prefetching algorithm, see section 6.5;

The IP1 and IP2 algorithms require the use of clustering information from the C3-GP clustering algorithm, to classify pages as 'resident' and 'non-resident'. C3-GP divides the database into two regions, 'hot' and 'cold'. IP1 and IP2 classify pages occurring in C3-GP's hot region as 'resident' and the remaining pages as 'non-resident'. In our experiments we set a C3-GP hot region size of 90% size of memory[9]. HP1 and HP2 rank database pages in terms of frequency of page reference. Once ranked, HP1 and HP2 classify the first *MEM_RES_PAGES* pages as being 'resident', where *MEM_RES_PAGES* multiplied by page size equals 50% of the memory size. HP1 and HP2 classify the remaining pages as being 'non-resident'.

In each experiment[10] the prefetch threshold[11] was set to 0.9 for every prefetch algorithm. We have tested different settings (at 0.1 increments) for each prefetching algorithm and found that the best setting was 0.9 in every case.

The performance metric used to measure the prefetching algorithms is the ratio of prefetch IO stall time over demand paging IO stall time. This is an indication of the percentage of IO time that the prefetch algorithm was able to hide. The reason we used this metric instead of PQM to measure performance is that we are ultimately interested in reducing IO stall time. The PQM metric is used in this thesis as a guide to explain the intuitions that have led to the design of our prefetching algorithms.

It is important to note that the results that we report in this chapter are a preliminary simulation study in which the computational overheads normally associated with prefetching (such as predictor computation time, multi-threading costs, locking costs and data structure space costs) have been ignored. If these factors are incorporated into the results, the benefits of the various prefetching algorithms may diminish.

[9] This settings has been found to produce the best C3-GP clustering performance (see section 5.6).
[10] Except for the experiment where the prefetch threshold is varied.
[11] The minimum probability of success required before a page is prefetched.

6.7 Experimental Results

In this section we report the results of experiments conducted to compare the performance of three existing prefetching algorithms and four new algorithms produced using the PCCP framework.

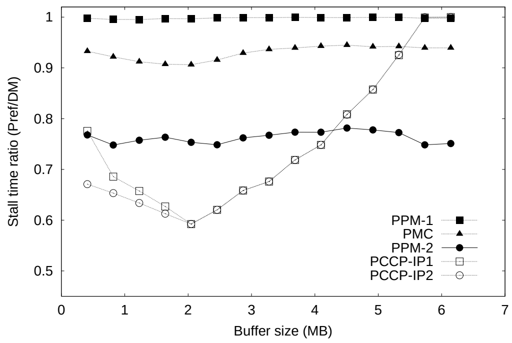
6.7.1 Varying Buffer Size

In this experiment we measure the effect of varying buffer size on the performance of the prefetching algorithms. Two sets of results are collected for this experiment, the first set uses the C3-GP clustering algorithm and the second set reports an average of the results from using three different clustering policies GGP, WGGP and no clustering. The effects of the three clustering algorithms individually are also reported in the appendix. Note PCCP-IP1 and PCCP-IP2 results are only shown for C3-GP results (figure 6.2 (a)) since PCCP-IP1 and PCCP-IP2 require clustering information from C3-GP to classify 'resident' and 'non-resident' pages.

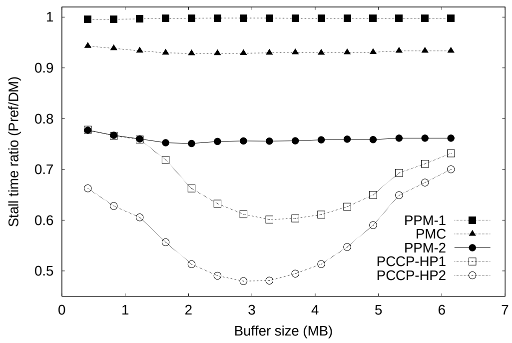
The prefetching results shown on figure 6.2 (a) depict the PCCP algorithms offering best performance for a range of buffer sizes (0.5MB to 4.2MB). The relatively poor performance of PCCP algorithms below 0.5 MB can be attributed to the small number of 'resident' page references between successive 'non-resident' page references. When the buffer size is small, the number of pages that can be kept in memory is also small, thus the probability of one 'non-resident' page reference followed straight after another (or only one or two 'resident' page references in between) is very high. This behaviour decreases prefetching time for PCCP algorithms. When the buffer size is large (at beyond 4 MB), almost the entire working set fits in memory. Thus almost all the pages in the working set are classified as 'resident' by the PCCP algorithms. Since PCCP algorithms only prefetch 'non-resident' pages and there are none of them in the working set, no prefetching is performed. Hence, the performance of PCCP algorithms rapidly degrades to that of demand fetching at these large buffer sizes.

Another observation from figure 6.2 (a) is that the two PCCP algorithms perform about the same. The reason for this is that C3-GP (the clustering algorithm used) takes all of the hot objects (which often has a large fan out) out of the cold pages and places then into the hot region which is then labeled as 'resident' by PCCP algorithms (thus no longer considered for prefetching). The absence of hot objects in cold pages (or 'non-resident' pages), means cold pages contain less objects with high fan out. The result is that the cold pages (pages used for prefetch prediction purposes) contain fewer paths of navigation. Under these simplified conditions for prediction, PCCP-IP1 performs almost as well as the more advanced PCCP-IP2.

There are two main observations that can be made from figure 6.2 (b). Firstly, the PCCP algorithms shown in figure 6.2 (b) exhibit a milder rate of performance degradation than figure 6.2 (a). The cause lies in the way the clustering policies work. The clustering policies GGP, WGGP and no clustering (used for 6.2 (b)), are not designed to produce pages of homogeneous heat whereas C3-GP is designed to produce pages of homogeneous heat. The result is that the clustering policies GGP, WGGP and no



(a) Using C3-GP clustering



(b) Using GGP, WGGP and no clustering

Figure 6.2: Results of varying buffer size. The results on the right report an average of the results from using three different clustering policies GGP, WGGP and no clustering. The appendix shows the results of using the clustering policies individually.

clustering need a larger buffer size to fit the entire working set in memory (the working set is spread across more pages). Hence, given the same buffer size, in figure 6.2 (b) PCCP classifies more pages that contain objects of the working set as 'non-resident'. Since PCCP algorithms only attempt to prefetch 'non-resident' pages, figure 6.2 (b) gives PCCP algorithms more opportunities for prefetching.

Secondly, in figure 6.2 (b) PCCP-HP2 outperforms PCCP-HP1 by a large margin. This contrasts with figure 6.2 (a) in which the two PCCP algorithms perform about the same. Unlike C3-GP, the clustering policies GGP, WGGP and no clustering do not extract hot objects from cold pages. The result is that 'non-resident' pages may contain hot objects (which often has a large fan out). In these conditions a 'non-resident' page (used by PCCP algorithms for prefetch prediction purposes) may contain many different paths of navigation. Hence, under these conditions, PCCP-HP2, which identifies navigational paths based on more historical reference information, can more accurately identify the current path of navigation when compared to PCCP-HP1.

The reason for the poor performance of PPM-1 is that it only uses the current state to predict the next state (SMC model). Furthermore, it stores prediction information at the page grain. The combination of the two drawbacks makes it very difficult to accurately distinguish between different paths of navigation early enough for prefetching. This problem is compounded by the rich schema and workloads (creating many different paths of navigation intersecting in a multitude of places) used in our experiments.

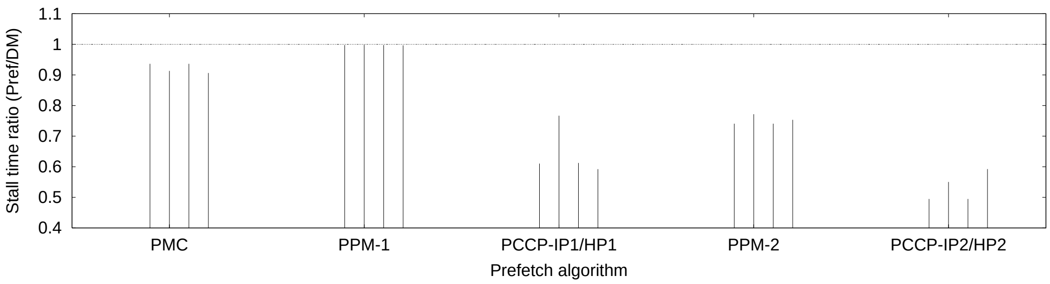


Figure 6.3: The impact of varying clustering algorithm. Each prefetching algorithm was tested against no clustering and three different clustering algorithms. The results are shown in the following order: no clustering; WGGP; GGP; and C3-GP.

6.7.2 Varying Clustering Algorithm

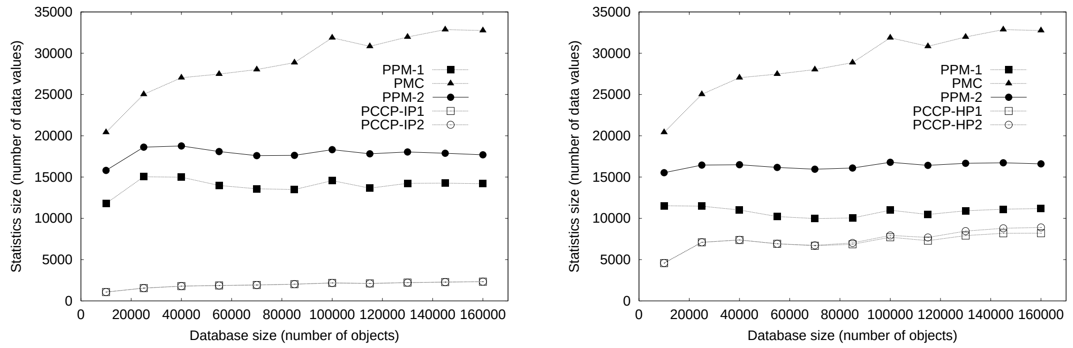
In this experiment, we examine the effect that varying clustering algorithms has on prefetching algorithm performance. The results are shown on figure 6.3. The buffer size was set to 2 MB. For each prefetching algorithm, the results of no clustering and three different clustering algorithms are reported in the following order: no clustering; the Wisconsin greedy graph partition algorithm (WGGP) [Tsangaris 1992]; the greedy graph partitioning algorithm (GGP) [Gerlhof et al. 1993]; and the C3-GP clustering algorithm. PCCP-IP1 and PCCP-IP2 prefetching algorithms are used when the C3-GP clustering algorithm is used. PCCP-HP1 and PCCP-HP2 are used for the remaining clustering policies.

The results show that the PCCP algorithms outperform existing prefetching algorithms for all clustering algorithms tested, including no clustering. This is an important result as it indicates that PCCP algorithms are likely to perform well given a wide variety of page-level access patterns. This is because clustering is responsible for object to page mappings which in turn determine page-level access patterns (the order pages are referenced and the duration of each page reference).

6.7.3 Statistics Storage Costs

In this experiment we examine the statistics storage requirements of the prefetching algorithms. The results show the number of statistics data values stored instead of the size of the data structures needed. The reason for this is that there are many different existing data structures which have various speed to space trade-offs[12], including some that limit statistics space consumption by flushing and rebuilding the data structures once a size limit has been reached. However, all of the data structures will benefit from a smaller number of data values.

[12]Data structures used for the prefetch algorithms tested require the index key to be a combination of a pair of ids, and thus simple data structures like arrays are precluded.



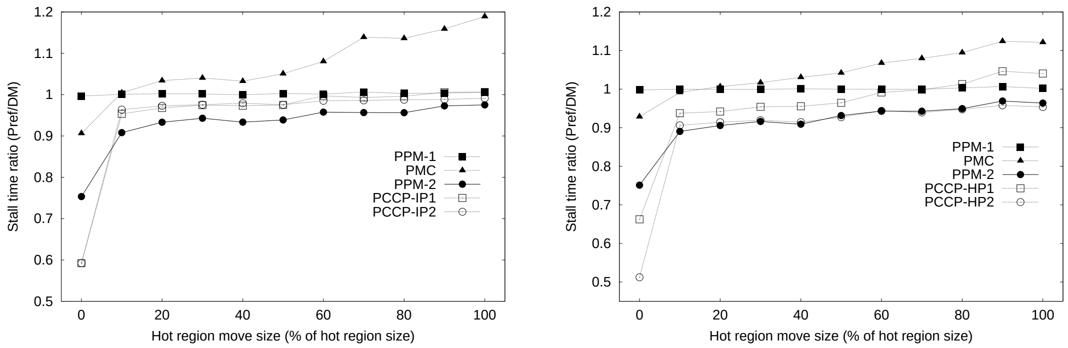
(a) Using C3-GP clustering

(b) Using GGP, WGGP and no clustering

Figure 6.4: Statistics storage cost results. The results on the right report an average of the results from using three different clustering policies GGP, WGGP and no clustering. The appendix shows the results of using the clustering policies individually.

The results are shown on figure 6.4. The buffer size used in this experiment was 2 MB. PCCP algorithms require the least space for storing data values. PCCP derives its cost savings mainly from restricting the storage of statistics to only 'non-resident' pages. In addition, the low statistics storage requirements of path conscious prefetching (storing short feature points) also helps to reduce the storage costs of the PCCP algorithms. These results show that path and cache conscious information (used by PCCP algorithms) can be stored efficiently. In fact, PCCP algorithms use only between 10% and 80% of the space needed by the first order PPM-1 coarse *page* grained prefetching algorithm and only between 8% and 53% of the second order PPM-2 algorithm.

A surprising result is that PCCP-IP2 and PCCP-HP2 store around the same number of data values as their single page counterparts (PCCP-IP1 and PCCP-HP1). For the purposes of explaining this behaviour let us assume *n* navigations passing through an entry object (first object in a page to be referenced) goes to *n* different target pages (next page referenced). Further assume the *n* navigations each originate from different previous page entry objects. Under these conditions, PCCP-IP2 and PCCP-HP2 store the same number of data values as their single page counterparts. We now explain why this type of navigational pattern occurs often in our experiments. It is due to a combination of the cache conscious feature of PCCP algorithms and trace characteristics. PCCP's cache conscious feature excludes 'resident' pages (pages more likely to contain hot objects) from prefetching statistics. Due to trace characteristics it is often the hot objects that have high fan outs. Thus the effect is hot objects that have high fan out (which produces large number of diverging paths of navigation) are excluded from prefetching statistics.



(a) Using C3-GP clustering

(b) Using GGP, WGGP, and no clustering

Figure 6.5: Results of varying training skew. The results on the right report an average of the results from using three different clustering policies GGP, WGGP and no clustering. The appendix shows the results of using the clustering policies individually.

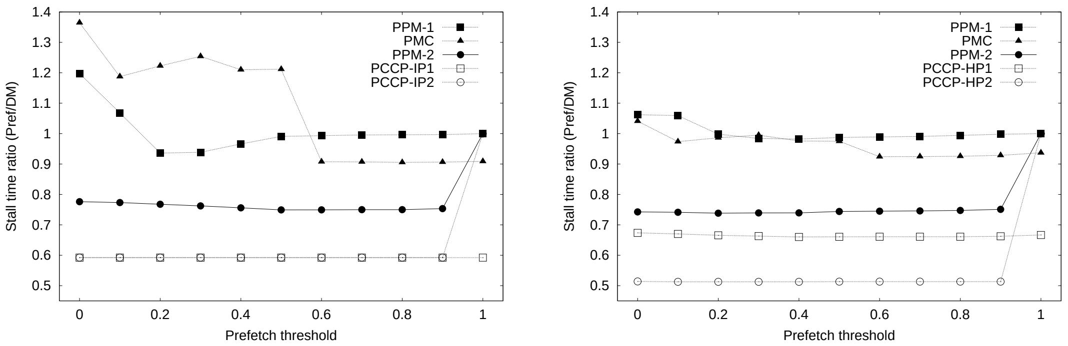
6.7.4 Varying Training Skew

Until now, all of the experiments involved running the same set of transactions for the clustering training step, prefetching training step and evaluation step. In contrast, this experiment explores the effect of running a different set of transactions during the prefetching training step and evaluation step, thus testing the sensitivity of the various prefetching algorithms to changes in access patterns. This is achieved by moving the hot region. The numbers on the x-axis of figure 6.5 show by how much the database hot region is moved. For example a value of 20% means there is 80% overlap between the hot region used for the prefetching training step and evaluation step. The hot region was set to 3% of the database size. The buffer size was set to 2 MB.

The results of this experiment are shown on figure 6.5. When training skew is below 10%, PCCP algorithms offer the best performance. However, above 10% skew, PCCP algorithms lose their advantage. The reason for this is that when skew is introduced, many trained paths of navigation no longer occur during the evaluation step. The result is that PCCP algorithms can rarely identify previously seen paths of navigation for prefetch prediction purposes. It is encouraging to note that PCCP-IP2 and PCCP-HP2 never perform worse than demand fetching. The reason is that PCCP-IP2 and PCCP-HP store path sensitive statistics. Using these statistics for prediction means close to exact path matches need to occur before prefetching is triggered. This limits the triggering of inaccurate prefetches.

6.7.5 Varying Prefetch Threshold

In this experiment we vary the prefetch threshold of each prefetching algorithm. The prefetch threshold is a user defined parameter that specifies the minimum probability



(a) Using C3-GP clustering (b) Using GGP, WGGP and no clustering

Figure 6.6: Results of varying the prefetch threshold. The results on the right report an average of the results from using three different clustering policies GGP, WGGP and no clustering. The appendix shows the results of using the clustering policies individually.

of reference required before a prefetch is allowed to occur. The buffer size is set to 2 MB.

The results are shown on figure 6.6. PPM-1 and PMC are sensitive to changes in the prefetch threshold, especially at low threshold values. In contrast, the PCCP algorithms and PPM-2 are very insensitive to prefetch threshold. The reason for this is that they keep more sensitive path information and thus can often identify situations where there is a *single* target page to prefetch.

6.7.6 Discussion

The results show that PCCP algorithms outperform existing prefetching algorithms in a variety of situations. In some situations PCCP algorithms outperform the existing first order page grained PPM-1 algorithm by as much as 53% of IO stall time and the existing object grain PMC algorithm by as much as 46% of IO stall time and finally the second order page grained PPM-2 algorithm by as much as 30% of IO stall time.

Two important pieces of evidence supporting the robustness of PCCP include: the varying clustering algorithm results; and changing prefetch threshold results. PCCP algorithms were found to outperform existing prefetch algorithms for all four clustering algorithms tested (one of which is no clustering). This result is of particular significance since clustering algorithms play a big part in determining page-level access patterns. This means that if a prefetching algorithm performs well using a variety of clustering algorithms, it is likely to perform well given a variety of different page-level access patterns. PCCP algorithms were also found to be insensitive to varying prefetch threshold. This lightens the load from users who aim to choose the best prefetch threshold value.

A key feature of PCCP is its low storage overhead. In our data structure size experiment we found that PCCP stores only between 10% and 80% the number of data values as the first order PPM-1 algorithm (a page grained prefetch algorithm), and only between 8% and 53% as much as the second order PPM-2 algorithm.

The training skew experiment showed that although large training skew degrades PCCP algorithm performance to about the same level as existing algorithms, it never performs worse than demand fetching. More experiments need to be done in this area to assess whether these results hold true under different conditions of training skew. However, these initial results are encouraging.

6.8 Conclusion

In this chapter we highlight the performance advantages of prefetching algorithms that incorporate synergies between prefetching and buffer replacement. To this end we describe the path and cache conscious prefetching framework (PCCP). Using PCCP, we produce four new prefetching algorithms. We compare PCCP against three highly competitive existing prefetching algorithms, PPM-1, PPM-2 and SP, and find that PCCP algorithms offer superior performance in a variety of situations.

Chapter 7

Conclusion

In this chapter we first present a summary of the main findings of this thesis. Secondly, we outline directions for future research and how this thesis can be used as a foundation. Finally, we discuss the key conclusions of this thesis.

7.1 Summary

The ever increasing demand for fast complex data storage and retrieval makes a strong case for OODBMSs' survival as an important database management technology. OODBMSs' good performance is derived from its ability to effectively handle navigational access of complex data. However, the ability for OODBMS to provide fast navigational access is conditioned on efficient main memory buffer management, which is made more important by the fact that disk IO performance improves at only 5-8% per year whereas CPU performance doubles approximately every 18 months.

We believe that the key to improving buffer management technique performance is to take a synergistic approach. Four buffer management techniques were involved in our study: static clustering; dynamic clustering; buffer replacement; and prefetching. In order to demonstrate the superiority of the synergistic approach to buffer management, we made general modifications to existing techniques. The modifications were made under the guiding principles of: *synergy*; *generality*; and *simplicity*. Following these guiding principles we developed three synergistic frameworks: opportunistic prioritised clustering framework (OPCF); cache conscious clustering framework (C3); path and cache conscious prefetching framework (PCCP). Each framework addresses the synergies between two different buffer management areas.

Chapter 2 laid the foundations for proper understanding of the synergy frameworks. This was done by first describing the OODBMS concepts important to this thesis and then defining the scope of this thesis within these concepts. Fundamental concepts such as the characteristics of object-oriented programming languages, object identity, and OODBMS functionality were described. In addition, various OODBMS architectural concepts and design alternatives were described along with the particular design alternatives chosen for this thesis. In this thesis we chose to explore storage management issues of the stand-alone peer-to-peer OODBMSs using page grained caching. The reasons for these decision were summarised in Section 2.5.

Chapter 3 served two purposes. First it further narrowed the scope of thesis by providing a more rigorous definition of the system model studied in the thesis. Second it defined a reference model and used it to provide a concise definition of the general problem addressed by this thesis. Third it presented the theoretical foundations (cost models and practical reference models) used throughout this thesis to explain the intuition behind the performance advantages of the various synergistic buffer management techniques.

In chapter 4 the opportunistic prioritised clustering framework (OPCF) was described. OPCF exploits the synergy between *static* and *dynamic clustering* algorithms to produce a framework that transforms static clustering algorithms into dynamic algorithms. OPCF is a general framework that creates dynamic clustering algorithms by placing opportunism and prioritisation into existing static clustering algorithms. In addition, application of the framework is straightforward. To demonstrate the strengths of OPCF we instantiated it for two existing contrasting static clustering algorithms. The algorithms we chose were the static probability ranking principles algorithm (PRP) and the static greedy graph partition algorithm (GGP). We performed extensive experiments and found that the dynamic algorithms produced, outperformed the existing highly competitive dynamic clustering algorithms DSTC and DRO in a variety of situations. The simplicity of the framework and the robustness of algorithms produced by it makes OPCF algorithms ideal candidates for inclusion in real OODBMS systems where workload conditions are likely to change with time.

Having addressed the synergy between static and dynamic clustering algorithms we moved on to *static clustering* and *buffer replacement* in Chapter 5. In that chapter we re-examined the objective upon which existing static clustering algorithms were designed. We observed that existing algorithms were designed to perform best when the buffer size is one page (where buffer replacement is not an issue) and consequently tries to confine traversals to the same page. Since most buffer sizes are larger than one page, we developed a framework for creating clustering algorithms designed for larger caches. Following the approach of the thesis, we made the framework simple to apply and general in terms of the ability to incorporate a wide range of existing algorithms. The results show that the cache conscious approach indeed produces better results for a wide variety of situations including: 10 different buffer replacement algorithms tested, varies buffer sizes, database hot region sizes, access probabilities, and various amounts of training skew. These results further confirm our thesis, namely that synergistic buffer management algorithms are both feasible to implement and outperform their non-synergistic counterparts.

In chapter 6 we introduced a framework for producing a new family of prefetching algorithms called PCCP. The framework exploits the synergy between prefetching and buffer replacement to produce prefetching algorithms that are cheap in terms of statistics usage and profitable in terms of the amount of overlap between CPU and IO. Like the previous frameworks, PCCP possesses both the simple and general property while being synergistic. We instantiated PCCP by creating four new prefetching algorithms from it. Two of the instantiated algorithms integrated clustering information to make prefetching decisions. The other two integrated page heat (frequency with which pages are referenced) information to make prefetching decision. We tested the performance of four algorithms generated by PCCP in a wide variety of situations, including: varying buffer size, clustering algorithm, statistics storage costs, training skew, and prefetching threshold. The results showed that PCCP algorithms offer best performance in most situations. This adds further evidence that much can be gained from developing simple synergistic buffer management techniques.

7.2 Future Work

Single Integrated Framework The synergistic frameworks developed in this thesis each exploit the synergies between two different buffer management techniques. However, we believe frameworks that exploit synergies between all four buffer management techniques would produce better results. Although this thesis does not propose such a framework it nonetheless has setup the foundations for one to be developed. One possible approach is to produce a new framework by building OPCF on top of C3. Such a framework would be able to produce cache conscious dynamic clustering algorithms. We can then use PCCP to produce prefetching algorithms that use clustering information from the combined OPCF and C3 framework. This is only one of many possible fully synergistic frameworks (frameworks that exploit synergies from all four buffer management techniques) that can be produced.

General peer-to-peer and client/server models As stated in section 2.4.2 the techniques developed in this thesis are only designed for a stand-alone single node of the peer-to-peer network model. However, the techniques can be extended to work for the general peer-to-peer and client/server network models. We will now discuss how each of the frameworks can be extended to work for the general peer-to-peer and client/server models. The core components of the OPCF framework (opportunism and prioritisation) can be re-used since OPCF is designed to work in an incremental way. However, care must be exercised to ensure update consistency among the dynamic clustering threads of different nodes. In the case of the C3 framework, different approaches need to be taken for the peer-to-peer and client/server network models. For the peer-to-peer case the C3 framework needs to cluster for a cache the size of the total memory of all the nodes instead of just one node.[1] For instance, the hot region size parameter of C3-GP would be set to total memory size of all nodes (in the case of low replication). For the client/server model, the size of the server cache can be used as the buffer size to cluster for. The PCCP algorithm needs to take network latency and remote disk accesses latency into consideration when deciding the value for the prefetch threshold.

[1] Assuming there is not much replication of cached pages among the different nodes. The algorithm needs to be adjusted when there is a high level of replication. This approach also assumes network latencies are much lower than IO latencies.

Experiments on real OODBMS The experiments in this thesis are conducted using a OODBMS simulator. Although the simulator provides good insights into algorithm performance through total IO and total IO stall time, it nonetheless is limited in its ability to simulate and measure all system components. Some system components that can be measured using real OODBMSs include buffer management computation time, multi-threading costs, locking costs and data structure space costs. Thus testing our algorithms on real OODBMS is an important area of future work. The state-of-the-art Platypus object store [He et al. 2000] is a good candidate for these experiments. This is because it is freely available for download and has a number of novel features that combine to make it one of the best performing (in terms of multi-threading costs, locking costs, data structure space, etc.) object stores available.

New algorithms created from our synergistic frameworks There is much remaining scope for creating new synergistic buffer management techniques from the frameworks developed in this thesis. For example, OPCF can be used to transform cache conscious clustering algorithms into dynamic clustering algorithms. New C3 algorithms can be developed by providing new definitions of regularity and new policies for dividing databases into homogeneous regularity regions. PCCP can be used to produce new prefetching algorithms that use new feature point selection policies and new definitions of 'resident' / 'non-resident' page metrics.

7.3 Conclusion

The main conclusion of this thesis 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.

Our guiding principles of *synergy*; *generality*; and *simplicity* used when designing our frameworks proved to be successful. The synergistic algorithms produced by the framework are easy and straightforward to implement, while providing good performance. In addition, the generality of the frameworks means new algorithms that possess the synergistic properties of the framework can be easily created.

The preliminary results of this thesis 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.

Appendix A

Additional PCCP Results

This appendix is intended as a supplement to the results reported in section 6.7. The results contained in this appendix is that of PCCP prefetching results when using the clustering policies of GGP, WGGP and no clustering.

A.1 Varying Buffer Size Experiment

Figure A.1 contains supplementary results for the experiment reported on section 6.7.1.

A.2 Statistics Storage Costs Experiment

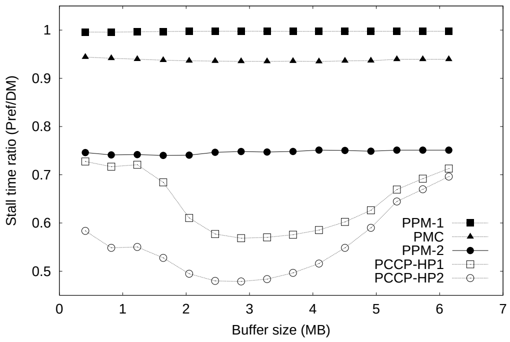
Figure A.2 contains supplementary results for the experiment reported on section 6.7.3.

A.3 Varying Training Skew Experiment

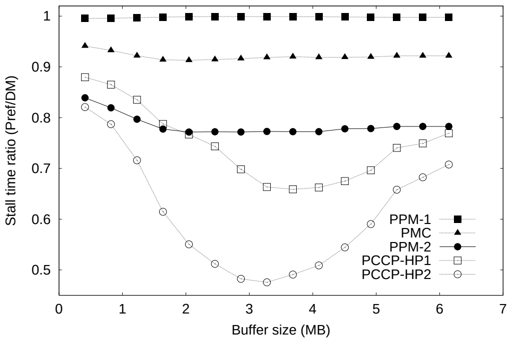
Figure A.3 contains supplementary results for the experiment reported on section 6.7.4.

A.4 Varying Prefetch Threshold Experiment

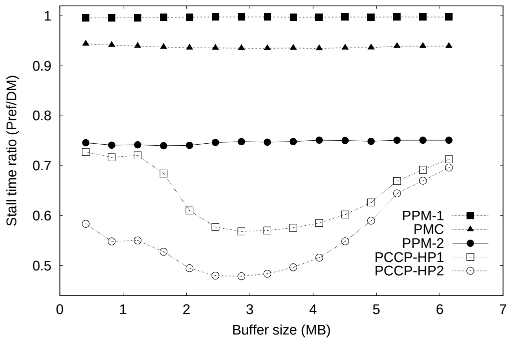
Figure A.4 contains supplementary results for the experiment reported on section 6.7.5.



(a) Using GGP clustering

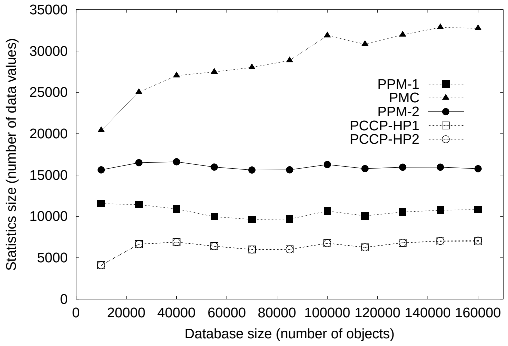


(b) Using WGGP clustering

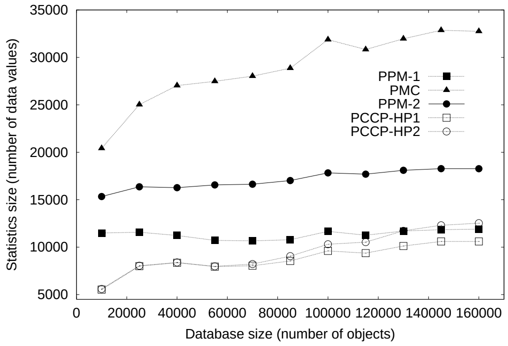


(c) Using no clustering

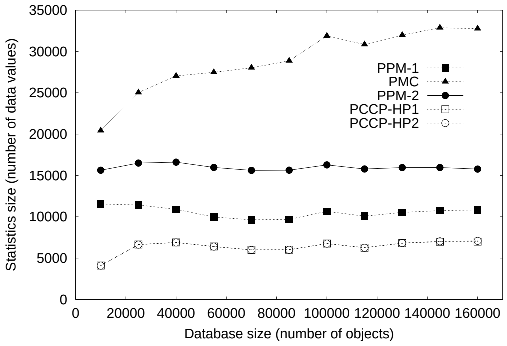
Figure A.1: Varying buffer size results.



(a) Using GGP clustering

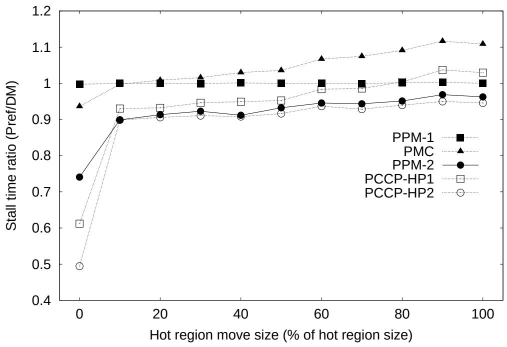


(b) Using WGGP clustering

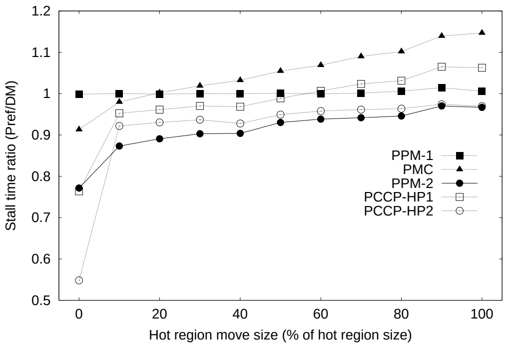


(c) Using no clustering

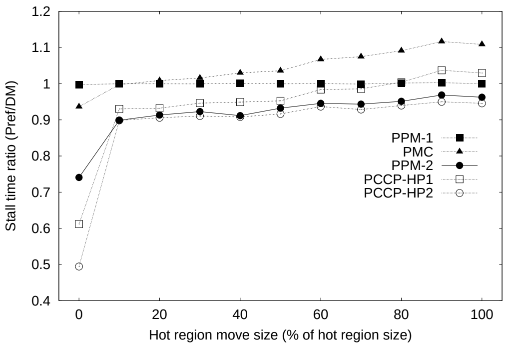
Figure A.2: Statistics storage size results.



(a) Using GGP clustering

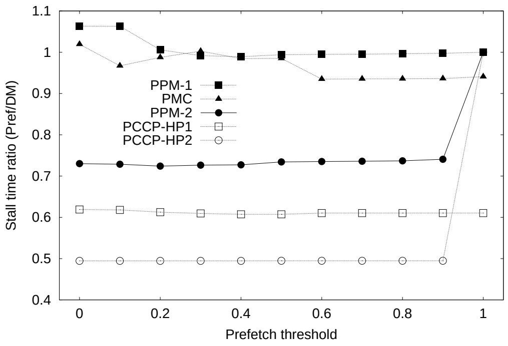


(b) Using WGGP clustering

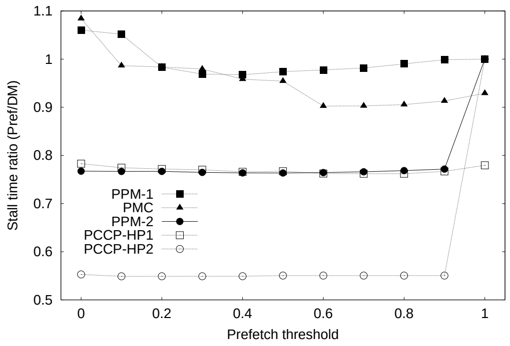


(c) Using no clustering

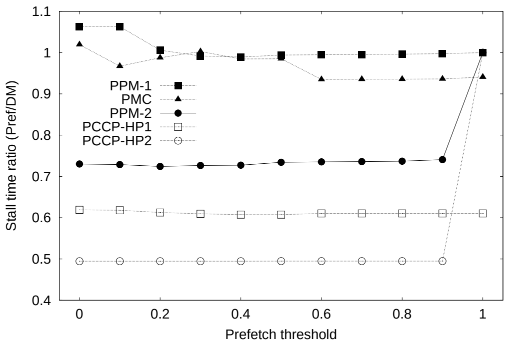
Figure A.3: Varying training skew results.



(a) Using GGP clustering



(b) Using WGGP clustering



(c) Using no clustering

Figure A.4: Varying prefetch threshold results.

Bibliography

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)

ARNOLD, A. O. 1978. *Probability, Statistics, and Queueing Theory, with Computer Scient Application*. Academic Press. (pp. 21, 22)

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)

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

BENZAKEN, V. AND DELOBEL, C. 1990. Enhaning performance in a persistent object store: Clustering strategies in o_2. Technical Report 50-90 (August), Altair. (p. 54)

BERNSTEIN, P. A., PAL, S., AND SHUTT, D. 1999. Context-based prefetching 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)

BULLAT, F. AND SCHNEIDER, M. 1996. 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), pp. 344–365. Springer. (pp. 3, 27, 28, 31, 32, 37, 44)

CAO, P., FELTEN, E. W., KARLIN, A. R., AND LI, K. 1995. A study of integrated prefetching and caching strategies. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1995)* (1995), pp. 188–197. (pp. 3, 72)

CAREY, M., DEWITT, D., AND NAUGHTON, J. 1993. The OO7 benchmark. *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1993)*. (p. 12)

CAREY, M. J., DEWITT, D. J., FRANK, D., GRAEFE, G., MURALIKRISHNA, M., RICHARDSON, J. E., AND SHEKITA, E. J. 1986. The architecture of the exodus extensible dbms. In *Proceedings of the International Conference on Very Large Databases (VLDB 1986)* (1986). (pp. 10, 11, 12)

CAREY, M. J., DEWITT, D. J., FRANKLIN, M. J., HALL, N. E., MCAULIFFE, M., NAUGHTON, J. F., SCHUH, D. T., SOLOMON, M. H., TAN, C. K., TSATALOS, O., WHITE, S., AND ZWILLING, M. J. 1994. Shoring up persistent applications. In *Proceedings of the International Conference on the Management of Data, (ACM SIGMOD 1994)* (Minneapolis, Minn, May 1994). (pp. 10, 12)

CAREY, M. J., FRANKLIN, M. J., LIVNY, M., AND SHEKITA, E. J. 1991. Data caching tradeoffs in client-server dbms architectures. In J. CLIFFORD AND R. KING Eds., *Proceedings of the International conference on the Management of Data (ACM SIGMOD 1991)* (1991), pp. 357–366. (p. 43)

CHANG, E. E. AND KATZ, R. H. 1989. Exploiting inheritance and structure semantics for effective clustering and buffering in an object-oriented DBMS. In J. CLIFFORD, B. G. LINDSAY, AND D. MAIER Eds., *Proceedings of the International Conference on Management of Data (ACM SIGMOD 1989), Portland, Oregon* (1989), pp. 348–357. ACM Press. (pp. 70, 71, 72)

CHEN, S., GIBBONS, P. B., AND MOWRY, T. C. 2001. Improving index performance through prefetching. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 2001)* (2001), pp. 235–246. (p. 1)

CUREWITZ, K. M., KRISHNAN, P., AND VITTER, J. S. 1993. Practical prefetching via data compression. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1993)* (Washington, DC, May 1993), pp. 43–53. (pp. 70, 72, 82)

DARMONT, J. 2000. Desp-c++:a discrete-event simulation package for c++. *Software Practice and Experience 30*, 1, 37–60. (p. 40)

DARMONT, J., FROMANTIN, C., REGNIER, S., GRUENWALD, L., AND SCHNEIDER, M. 2000. Dynamic clustering in object-oriented databases: An advocacy for simplicity. In *Proceedings of the International Symposium on Object and Databases*, Volume 1944 of *LNCS* (June 2000), pp. 71–85. (pp. 27, 28, 38, 44)

DARMONT, J., PETIT, B., AND SCHNEIDER, M. 1998. 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), pp. 326–340. LNCS Vol. 1377 (Springer). (pp. 41, 42)

DARMONT, J. AND SCHNEIDER, M. 1999. 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. (pp. 39, 80)

DENNING, P. J. 1968. The working set model of program behavior. *Commincations of ACM 11*, 5 (May), 323–333. (p. 56)

DEUX, O. 1991. The O_2 system. *Communications of ACM 34*, 10, 34–48. (pp. 10, 11, 12, 40)

DREW, P., KING, R., AND HUDSON, S. E. 1990. The performance and utility of the cactis implementation algorithms. In D. MCLEOD, R. SACKS-DAVIS, AND H.-J.

SCHEK Eds., *Proceedings of the International Conference on Very Large Databases (VLDB 1990)* (Brisbane, Queensland, Australia, 13–16 Aug. 1990), pp. 135–147. Morgan Kaufmann. (pp. 25, 26, 28, 33, 53, 54)

FRANKLIN, M. J., CAREY, M. J., AND LIVNY, M. 1993. Local disk caching for client-server database systems. In R. AGRAWAL, S. BAKER, AND D. A. BELL Eds., *Proceedings of the International Conference on Very Large Databases (VLDB 1993)* (1993), pp. 641–655. (p. 43)

GAY, J.-Y. AND GRUENWALD, L. 1997. A clustering technique for object oriented databases. In *Proceedings of the International Conference on Database and Expert Systems (DEXA 1997)*, Volume 1308 of *Lecture Notes in Computer Science* (September 1997), pp. 81–90. Springer. (pp. 27, 38)

GERLHOF, A., KEMPER, A., AND MOERKOTTE, G. 1996. On the cost of monitoring and reorganization of object bases for clustering. In *ACM SIGMOD Record*, Volume 25 (1996), pp. 28–33. (p. 25)

GERLHOF, C., KEMPER, A., KILGER, C., AND MOERKOTTE, G. 1993. Partition-based clustering in object bases: From theory to practice. *Proceedings of the International Conference on Foundations of Data Organisation and Algorithms (FODO 1993)*, 301–316. (pp. 25, 26, 28, 35, 36, 53, 54, 55, 60, 61, 81, 85)

GERLHOF, C. A. AND KEMPER, A. 1994. A multi-threaded architecture for prefetching in object bases. In *Proceedings of the International Conference on Extended Database Technology (EDBT 1994)* (1994), pp. 351–364. (pp. 3, 71)

GRAY, J. AND PUTZOLU, G. R. 1987. 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. (pp. 2, 43)

HAN, W., WHANG, K., MOON, Y., AND SONG, I. 2001. Prefetching based on the type-level access patterns in object-relational DBMSs. In *Proceedings of IEEE international Conference on Data Engineering (ICDE 2001)* (2001), pp. 651–660. (pp. 70, 72)

HE, Z., BLACKBURN, S. M., KIRBY, L., AND ZIGMAN, J. 2000. Platypus: Design and implementation of a flexible high performance object store. In *Proceedings of the International Workshop on Persistent Object Systems (POS 2000)* (2000), pp. 100–124. (pp. 10, 11, 12, 94)

HE, Z. AND DARMONT, J. 2003. Dynamic object evalutaion framework (DOEF). In *Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA 2003)* (2003), pp. 662–671. (pp. 6, 41, 43)

HE, Z. AND MARQUEZ, A. 2001. Cache conscious clustering C3. In *Proceedings of the International Database and Expert Systems Applications Conference (DEXA 2001)* (Munich, Germany, September 2001), pp. 815–825. (p. 6)

HE, Z., MARQUEZ, A., AND BLACKBURN, S. 2000. Opportunistic prioritised clustering framework (OPCF). In *Proceedings of the International Symposium on Object and*

Databases, Volume 1944 of *LNCS* (June 2000), pp. 86–100. (p. 6)

HUDSON, E. AND KING, R. 1989. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. In *ACM Transactions on Database Systems* (September 1989), pp. 291–321. (p. 28)

JOHNSON, T. AND SHASHA, D. 1994. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the International Conference on Very Large Databases (VLDB 1994)* (1994), pp. 439–450. (pp. 54, 55)

JOSEPH, M. 1970. An analysis of paging and program behaviour. *The Computer Journal 13*, 1, 48–54. (pp. 71, 72)

KEMPER, A. AND KOSSMANN, D. 1994. Dual-buffering strategies in object bases. In J. B. BOCCA, M. JARKE, AND C. ZANIOLO Eds., *Proceedings of the International Conference on Very Large Databases (VLDB 1994), Santiago de Chile, Chile* (1994), pp. 427–438. Morgan Kaufmann. (p. 11)

KERNIGHAN, B. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 291–307. (p. 35)

KNAFLA, N. 1997. A prefetching technique for object-oriented databases. In *Advances in Databases, 15th British National Conf. on Databases* (London, United Kingdom, 1997), pp. 154–168. Springer-Verlag. (pp. 70, 71, 72)

KNAFLA, N. 1998. Analysing object relationships to predict page access for prefetching. In *Proceedings of the International Workshop on Persistent Object Systems (POS 1998)* (Tiburon, California, 1998), pp. 160–170. Morgan Kaufman. (pp. 70, 71, 72, 82)

LAKHAMRAJU, M., RASTOGI, R., SESHADRI, S., AND SUDARSHAN, S. 2000. On-line reorganisation in object databases. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 2000)* (May 2000). (p. 8)

LAMB, C., LANDIS, G., ORENSTEIN, J., AND WEINREB, D. 1991. The objectStore database system. *Communications of ACM 34*, 10, 50–63. (pp. 10, 11, 12)

LEE, D., CHOI, J., KIM, J.-H., NOH, S. H., MIN, S. L., CHO, Y., AND KIM, C. S. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1999)* (1999), pp. 134–143. (pp. 54, 55)

LISKOV, B., ADYA, A., CASTRO, M., MAY, M., GHEMAWAT, S., GRUBER, R., MAHESHWARI, U., MYERS, A. C., AND SHRIRA, L. 1996. Safe and efficient sharing of persistent objects in thor. In *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1996)* (1996), pp. 318–329. (pp. 11, 12, 71)

MCIVER, W. J. J. AND KING, R. 1994. Self-adaptive, on-line reclustering of complex object data. In R. T. SNODGRASS AND M. WINSLETT Eds., *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1994), Minneapolis, Minnesota* (1994), pp. 407–418. ACM Press. (pp. 27, 28)

NICOLA, V. F., DAN, A., AND DIAS, D. M. 1992. Analysis of the generalized clock buffer replacement scheme for database transaction processing. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1992)* (1992), pp. 35–46. (pp. 54, 55, 63)

O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. 1993. The LRU-K page replacement algorithm for database disk buffering. In P. BUNEMAN AND S. JAJODIA Eds., *Proceedings of the International Conference on the Management of Data (ACM SIGMOD 1993) , Washington, D.C.* (1993), pp. 297–306. ACM Press. (pp. 54, 55, 63)

PALMER, M. AND ZDONIK, S. B. 1991. Fido: A cache that learns to fetch. In *Proceedings of the International Conference on Very Large Databases (VLDB 1991)* (Septmeber 1991), pp. 255–264. (pp. 70, 71, 72)

RAO, J. AND ROSS, K. A. 1999. Cache conscious indexing for decision-support in main memory. In *Proceedings of 25th VLDB Conference* (Sept. 1999), pp. 78–89. (p. 1)

RAO, J. AND ROSS, K. A. 2000. Making B^+-trees cache conscious in main memory. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 2000)* (May 2000), pp. 475–486. (p. 1)

ROBINSON, J. AND DEVARAKONDA, N. 1990. Data cache management using frequency-based replacement. In *Proceedings of the International Conference on the Measurement and Modeling of Computer Systems, (ACM SIGMETRICS 1990)* (1990), pp. 134–142. (pp. 54, 55)

SINGHAL, V., KAKKAD, S. V., AND WILSON, P. R. 1992. Texas: An efficient, portable persistent store. In *Proceedings of the International Workshop on Persistent Object Systems (POS 1992)* (1992), pp. 11–33. (p. 40)

STAMOS, J. 1984. Static grouping of small objects to enhance performance of a paged virtual memory. In *ACM Transactions on Computer Systems*, Volume 2 (May 1984), pp. 155–180. (p. 54)

TSANGARIS, E.-M. M. 1992. *Principles of Static Clustering For Object Oriented Databases*. PhD thesis, University of Wisconsin-Madison. (pp. 25, 26, 28, 33, 35, 53, 54, 55, 56, 57, 60, 61, 81, 85)

TSANGARIS, M. M. AND NAUGHTON, J. F. 1991. 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. (p. 32)

TSANGARIS, M. M. AND NAUGHTON, J. F. 1992. 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. (pp. 32, 33, 42)

WIETRZYK, V. S. AND ORGUN, M. A. 1999. Dynamic reorganisation of object databases. In *Proceedings of the International Database Engineering and Applications Symposium* (August 1999). IEEE Computer Society. (pp. 26, 27, 28)

WILSON, P. R., JOHNSTONE, M. S., NEELY, M., AND BOLES, D. 1995. Dynamic storage allocation: A survey and critical review. In *International Workshop on Memory Management* (Kinross, Scotland, UK, Sept 1995). (p. 41)

YUE, P. C. AND WONG, C. K. 1973. On the optimality of the probability ranking scheme in storage applications. *JACM 20*, 4 (October), 624–633. (p. 56)

Index