A STOCHASTIC APPROACH FOR CLUSTERING IN OBJECT BASES

by

Manolis M. Tsangaris and Jeffrey F. Naughton

Computer Sciences Technical Report #1017

May 1991

A Stochastic Approach for Clustering in Object Bases* †

Manolis M. Tsangaris and Jeffrey F. Naughton
{mt,naughton}@cs.wisc.edu
Department of Computer Sciences
University of Wisconsin–Madison

May 1, 1991

Abstract

Object clustering has long been recognized as important to the performance of object bases, but in most work to date, it is not clear exactly what is being optimized or how optimal are the solutions obtained. We give a rigorous treatment of a fundamental problem in clustering: given an object base and a probabilistic description of the expected access patterns, what is an optimal object clustering, and how can this optimal clustering be found or approximated? We present a system model for the clustering problem and discuss two models for access patterns in the system. For the first, exact optimal clustering strategies can be found; for the second, we show that the problem is NP-complete, but that it is an instance of a well-studied graph partitioning problem. We propose a new clustering algorithm based upon Kernighan's heuristic graph partitioning algorithm, and present an experimental comparison of this new clustering algorithm with several previously proposed clustering algorithms.

*This work was supported by an NSF grant number IRI–8909795, TOPAZ DCR 8521228, and partially by a NATO fellowship.

†A shorter version of this paper will appear in SIGMOD 1991.

To the memory of Emmanuel M. Tsangaris

1 Introduction

It is widely acknowledged that good object clustering is critical to the performance of object-oriented database systems. However, in most work to date on object clustering, it is not clear exactly what is being optimized, or how optimal the proposed solutions are. In this paper, we give a rigorous treatment of what we believe is the fundamental problem in clustering: given an object base and a probabilistic description of the expected access patterns over the object base, what constitutes an optimal object clustering, and how can this optimal clustering strategy be found or approximated?

By focusing on this underlying problem we are omitting many commonly considered questions that arise when implementing a clustering strategy in a running system. For example, we do not consider how the description of the expected access patterns for the database is obtained. Many techniques for this have been proposed in the literature, including maintaining usage statistics, using programmer hints, and performing data-flow analysis of the methods and type hierarchy of the system. Similarly, we do not explicitly consider the "granularity" at which the clustering is performed. The clustering could be per type, or per instance, or per method (where the objects touched by a method are treated as a single, composite object) or even some hybrid of these.

By omitting these questions from consideration we do not wish to imply that they are unimportant. To the contrary, we consider good answers to these questions (and others) to be an essential component of the clustering strategy of an OODBMS. However, the fundamental question of what a "good clustering strategy" means underlies all these other questions, hence is worthy of study in its own right.

Previous work on object clustering can be divided into several categories. Methods that use programmer hints (E and EXODUS [RC88], Semantic Clustering [SS90]) rely on the skill of the programmer and the programmer's understanding of the problem. Syntactic methods (PS-Algol [CAC+84], LOOM-Smalltalk [Sta84]) determine a clustering strategy based solely upon the static structure of the object base. Dynamic methods (Cactis [HK89], ObServer [HZ87]) gather statistics about reference patterns, and try to find a "good" clustering based upon these statistics, while still other techniques (O_2 [BD90], WDFS in LOOM [Sta84]) can best be thought of as a hybrid of the syntactic and dynamic methods. The clustering criteria used in each of these types of methods are heuristics whose effect on the performance of the system is not known with any certainty. For this reason, evaluation of clustering strategies has relied upon either intuitive arguments or simulations, that mix the effects of the access stream, object base, memory size, and caching policy with the effect of the clustering strategy.

The following simple example illustrates why the clustering problem requires careful study. Suppose we have an object base that consists of six objects, which we will designate s, i, m, p, l, and e. Suppose also that there are pointers from s to i, from i to s and m, from m to p, from p to m and l, and finally from l to e and from e to l (this structure is illustrated by the solid arrows in Figure 1). Furthermore, assume that we can store any three of these objects in one disk page, and that our system can cache one disk page in main memory at any given time.

Next, suppose that when object s is referenced, there is an overwhelming tendency to reference

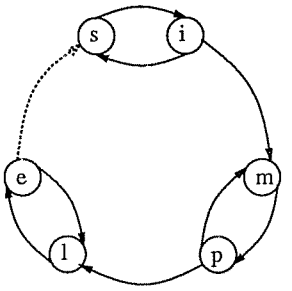


Figure 1: The "simple" example

object i next (say, 99 times out of 100). Also, suppose that if i is referenced, there is an overwhelming chance of referencing s next (again, say 99 times out of 100). Similarly, assume that from m we reference p 99 times out of 100, and from p we reference m 99 times out of 100. Finally, assume that l and e are related in the same way (99 times out of 100, when referencing one of the two, the other is referenced next). Given this information, a probable reference string might be $w = (si)^{99}(mp)^{99}(le)^{99}$.

Now consider two clustering strategies: $C_1 = \{[sim], [ple]\}$, and $C_2 = \{[si], [mp], [le]\}$. Under most of the clustering criteria we have seen proposed to date, C_1 is "optimal." In particular, C_1 minimizes the space occupied by the database; it minimizes the expected loading time for the database; it minimizes the number of pointers that cross page boundaries; for the reference string w given above, no object is ever brought into memory but not used at some point in the future. However, w will produce 101 page faults under clustering strategy C_1, while it will produce only 3 page faults under clustering strategy C_2.

In this paper we present a system model for the clustering problem and discuss two rigorous models for access patterns in the system. For the first, the *Independent Identical Distribution* (IID) model, exact optimal clustering strategies can be found. While the IID model is too simple to model detailed client access patterns, as we shall demonstrate, it can be a useful tool in designing clustering strategies for client-server environments. For the second, more powerful access model, the *Simple Markov Chain Model*, we show that the clustering problem is NP-complete; however, in this case, we show that the clustering problem is an instance of a well-studied graph partitioning problem for which good heuristic solutions are known. We propose a new clustering algorithm based upon Kernighan's heuristic graph partitioning algorithm, and present an experimental comparison of this new algorithm with previously proposed clustering algorithms.

2 Defining Clustering

In giving a first cut definition for clustering, we propose a simple client-server system model for object bases. The main points of this model are the client request stream and the cost of client-server interactions. Now the clustering can be formulated as an optimization problem.

2.1 The Client–Server model

In most Persistent Object Base (POB) systems there is a server entity that implements some object abstraction for its clients. We will make the standard assumption that all existing objects are denoted by a positive integer identifier and belong to a *fixed object space* S. The client makes requests to access objects, represented by the infinite sequence X_n, the *(client) access request stream*:

$$X_n = x_1, x_2 ... x_n ..., \ (x_i \in S)$$

At every time unit[1] n, the server must *return* to the client the requested object x_n. Since, clustering is closely related to the storage and access aspects of the objects, we further assume that every object $s \in S$ is assigned to some unit of storage $f = D(s)$, called *frame* (typically, a frame is a disk page). All frames belong to a *fixed frame space* F (denoted by a subset of the positive integers). We will use the notation f_i to indicate the set of all objects assigned to frame i. In this paper we assume that no object is larger than a frame. The set–to–point $(N \rightarrow 1)$ mapping $S \stackrel{D}{\rightarrow} F$ defines completely the above assignment.

For convenience we will sometimes view D as a two step mapping:

- A partition of S; $D_\pi : S \rightarrow F$
- A permutation of frames; $D_s : F \rightarrow F$

Consequently every such mapping D can be expressed as: $D(\cdot) = D_s(D_\pi(\cdot))$. For example, in a clustering scheme, S may be partitioned according to the types of the objects first. Then, the permutation D_s may sort the resulting frames with respect to their total access probability, arrange them in descending order, and place them on disk cylinders with that order.

2.2 The access request stream

We will assume that some statistical properties of the client access request stream X_n are known. The simplest way to describe X_n is as a sequence of independent and identically distributed random variables *(IID model)*. Their common value domain is the set of objects (S) and their distribution is defined by a probability vector $\vec{\pi}_s$:

$$\pi_s(y) \equiv Pr(x_n = y), \ \forall y \in S, \ n = 0, 1, ...$$
$$\text{and} \ \ \textstyle\sum_{y \in S} \pi_s(y) = 1$$

Obviously, this model captures no serial dependency information. An IID stream may contain any object reference y at any time with a fixed probability $\pi_s(y)$ independent of previous requests. To describe serial dependencies a more complex stochastic model must be used. The simplest such model is the *SMC (Simple Markov Chain) model*, with state space equal to the set S and a known transition probability matrix:

$$P_s(x, y) \equiv Pr(x_{n+1} = y | x_n = x), \ \forall x, y \in S$$

[1]The notion of time here corresponds to the sequence of requests and not directly to the elapsed time.

We will further assume that in the case of an SMC model, the state space S contains only one closed and irreducible set of recurrent states. That is, in finite time the client may request any object after any other object with positive probability. For this reason there exists a unique stationary probability distribution vector $\vec{\pi}_s$ corresponding to P_s, the solution of the matrix equation:

$$P_s \vec{\pi}_s = \vec{\pi}_s$$

2.3 Client-Server interaction

There are at least two possibilities for the client-server (c-s) communication. In object level communication, object is the granularity of interactions. At every time n the client requests the object x_n from the server. Then, the server retrieves the frame $f_n = D(x_n)$ from the secondary storage, it extracts the object x_n, and finally returns it to the client. Obviously *every* object access will require a c-s interaction.

Alternatively, frames can be the granularity of interactions. In frame level communication, at time n the client requests the frame $f_n = D(x_n)$. After receiving the frame f_n from the server, it is entitled to access not only x_n but any other object $y \in f_n$ as well. Presumably, the client must be able to *store* a whole frame, but subsequent consecutive accesses to the same frame will require no c-s interactions.

2.4 Cost of interactions

If c-s interactions were free, the clustering problem would not be interesting at all! In reality, each time a request is sent to the server the client is suspended and any computation it performs is delayed. Additionally, the server will spend some amount of resources to satisfy the request. We will assume that every c-s interaction has a cost that only depends on the previous request. So each object x_n fetch[2] costs $Q(x_{n-1}, x_n)$. As a result, the total cost T_n until time n is:

$$T_n = \begin{cases} T_{n-1} + Q(x_{n-1}, x_n) & n > 0, \\ 0 & n = 0 \end{cases}$$

Since all $Q(i,j)$ are positive (they represent cost), $lim_{n\to\infty} T_n = \infty$, and it makes more sense to consider the average cost $\bar{T}_n = \frac{T_n}{n}$ instead. $\bar{T}_n$ is just the mean of a sum of infinite random variables, and by virtue of the law of large numbers and assuming ergodicity[3] of the x_n's, it converges to the expected value of the random variable Q:

$$T = \lim_{n\to\infty} \bar{T}_n = E(Q(x_{n-1}, x_n))$$

A similar cost formula has been proposed elsewhere [YW73]. The definition of T is general enough to approximate many real world situations like disk accesses. Of course, if the client architecture is more complicated, for example if a server cache is used, then Q will depend not only on the last frame but on a number of previously requested frames.

[2] Or frame f_n fetch — whatever is appropriate depending on the style of interactions.

[3] True for the ID and SMC models.

2.5 Formulation of Clustering

Now we can formulate clustering as an optimization problem. Given a:

- statistical description of the client access stream X_n, and
- a frame access cost formula Q for the client-server interactions,

find the mapping $D : S \rightarrow F$ such that:

- the average cost T is minimized, while
- there are at most L objects per frame[4] and
- other additional constraints are satisfied.

The additional constraints can be arbitrary restrictions on where and how objects can be assigned. Such constraint would be to restricting placement of objects to frames because of their types or their sizes. Clustering in the presense of additional constraints can be very hard. For example the (frame) space minimization problem under variable size objects and no regards to the request stream, is NP–complete (the knapsack problem). In this paper we will ignore those additional constraints so we can concentrate on the rest of the problem.

For ease of presentation, the above simple model has ignored many details of the actual POB architectures. A more realistic model will be introduced in Section 3 where results from the simple model will be used.

2.6 Optimal Clustering

In this section, we first give solutions for optimal clustering using simple access models and relatively simple cost formulas. Next, we present some optimal clustering solutions for minimizing the expected cost under more complex cost metrics. In all cases object level c-s interaction is assumed.

The simplest access model is the IID. The simplest frame access cost formula is the uniform fixed cost one:

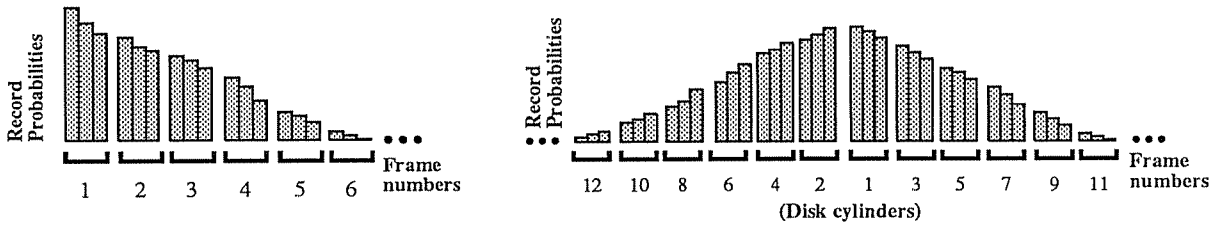
$$Q(x_{n-1}, x_n) = \alpha \ (1 - \delta_{x_{n-1}, x_n}) \tag{1}$$

where α is a non-negative constant and $\delta_{i,j} = 1 : i = j$, $\delta_{i,j} = 0 : i \neq j$. Because of buffering, there is no cost if the same object is requested again.

The *optimal* clustering scheme in the above case due to [YW73], is known as the probability ranking scheme. Its partition component D_π involves sorting the objects in descending order of their absolute probability $\vec{\pi}_s$ and successively assigning them to frames in that order as in Figure 2a, where objects with similar temperature[5] are assigned to the same frame as much as possible. The permutation component is the identity $(D_s(f) = f)$.

[4] If $L = \infty$ the problem has a trivial solution: assign all objects to one frame.

[5] The term *temperature* of an object is sometimes used instead of the term *absolute access probability* of an object for the IID case, or *stationary probability* of an object for the SMC case.



a: Probability Rankning **b: Organ Pipe Arrangement**

Figure 2: Two optimal clustering schemes

Next, one can change the cost formula Q, so, while it is still simple, it models the disk geometry characteristics more closely. In this context the frame corresponds to a disk block or disk cylinder. The cost of each object access corresponds to the cost of its frame, and the frame cost depends on the current position of the disk head. More specifically the cost is a function of the absolute *distance* from the last request:

$$Q(f_n, f_{n+1}) = c(|f_n - f_{n+1}|) \tag{2}$$

Additionally, some monotonicity restrictions to the function $c(n)$ exist, so that the longer the head travels the greater the cost is:

$$c(0) \leq c(n) \leq c(m),\ 0 \leq n \leq m$$

The optimal clustering can still be found (see [YW73], [GS73] and [Won83]). The partition component of the optimal mapping remains the same as before (probability ranking), but the permutation is not the identity any more. The resulting frames are sorted in descending order, according to the sum of the probabilities of the objects they contain, and placed on disk as in Figure 2b:

$$2n \ldots 6, 4, 2, 1, 3, 5 \ldots 2n+1$$

This clustering scheme is known as the *organ pipe arrangement.* In this arrangement, the hottest frames are placed in the middle of the disk. The two next to the hottest frame are placed adjacent to it and the process is repeated until all the frames have been assigned.

An adaptive system described in [VC90], uses the above method to dynamically permute disk blocks, so that the average observed access time is minimized. During an observation period, disk block accesses due to the Unix filesystem are measured, and an IID model is estimated from counting the accesses. Finally, a new permutation is computed and implemented if necessary. Substantial performance improvements have been observed in that system, and the average seek time was reduced by 40 – 50 %.

The solution for the SMC case is hard, especially for the disk cost formula (Equation 2). Fortunately, as we will show in Section 4, with a more realistic system model it becomes clear that it is not important to solve this problem with the complex cost formula.

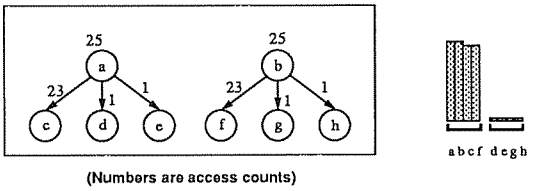


Figure 3: Clustering using probability ranking

Clearly, IID-probability ranking clustering can be better than purely syntactic clustering. IID clustering considers usage statistics, whereas syntactic methods use structural data only. But this data alone cannot reveal how the object structure will be used. In the example presented in Figure 3, syntactic methods will fail to place the hottest objects $\{a, b, c, f\}$ together because of their structural independence, however the probability ranking method will do so.

On the other hand, probability ranking will fail in the example of Figure 1. It has no way to distinguish between $\{s, i, m, p, l, e\}$, since, under the IID model, all objects have the same access probability. An SMC model can capture dependencies like $s \leftrightarrow i$, and as we will see next, it will produce the optimal partition C_2.

3 A more realistic model

The system model we have presented so far is somewhat simple. Usually, a POB system serves more than one client at the time. It also uses various levels of caches to increase the performance. In this section we discuss the effect of multiple clients and caching on a client-server system, and revise the system model appropriately. As we will observe, the revised system model is still subject to clustering optimizations but now with different objectives.

3.1 Adding multiple clients

The model we have presented in Section 2.1 can be enhanced by adding multiple clients and queueing of their requests at the server. Then, what are the statistical characteristics of the combined request stream (global request stream) of all clients?

We make the standard assumption that all client requests are independent and have similar behavior. In other words, all their (local) request streams follow the same statistical model. Then, the statistical characteristics of the combined request stream depend a lot on the queueing policy used (FIFO, Priority based, etc.) and on the load of the system [All78]. The heavier the load of the system, the more random the global request stream is. Then, in moderately to highly loaded multi-client system the global request stream loses its serial characteristics, and can be described adequately by an IID model. The probabilities of that model, are equal to the common (stationary) probabilities of the local request models.

This observation obviously affects the design of clustering. No matter how serially dependent the local requests are, queueing *de–serializes* the global request stream. In such an environment, the frame request cost depends more on the load of the server and less on the cost of retrieving

frames from the secondary storage. The load is clearly a dynamically changing parameter, and therefore, static clustering decisions cannot depend on it.

For these reasons, clustering in multi-client environment must have different objectives than in the single-client case. Globally optimizing for dynamic frame cost is out of the question. One standard way to attack the problem is deciding about each system component independently. Partitioning can be done depending on the local client stream characteristics, while frame permutation can be based upon the characteristics of the global request stream. This idea is developed further in Section 3.4.

3.2 Adding client caching

A primitive cache has already been introduced to the simple system model. In Section 2.3 a cache of size one frame was used to hold a requested frame. Subsequent consecutive references to the same frame are satisfied from the cache. Generalizing that cache is an obvious step.

The client frame cache sits between the client and the server, and is modeled as a set of frames C_n that changes with time, together with some cache replacement strategy. The cache contents depend on the previous requests and the cache replacement policy used. A cache miss because of x_n causes a client—server interaction. The frame $f_n = D_s(x_n)$ is requested from the server and it is stored back in the cache. If a replacement decision is needed, a cache frame is chosen and replaced by f_n.

It is very important to observe that both the frame cache and the server "see" a different access stream than the one the client originally produces (X_n). The server receives frame access requests that correspond to cache misses, and the cache does not see object requests, but frame requests that correspond to the mapped sequence X_n. A "bad" clustering strategy may destroy the locality of X_n. On the other hand, a "good" clustering strategy can enhance the locality of X_n.

In this environment we will assume that the client access cost is 0 when we have a cache hit, and 1 when a cache miss occurs. As a result, the total cost is proportional to the number of cache misses, and minimizing it is equivalent to minimizing the number of cache misses. Unfortunately, cache misses depend not only on the request stream but on cache management as well. This implies that such a clustering strategy *will depend much on the cache management policy.*

Having to deal with the cache policy makes clustering an even more difficult problem. After all, there is no easy way to describe all possible cache policies or to decide in advance which one would be the best. For this reason, instead of trying to minimize misses for caches of some specific type, clustering should attempt to enhance the locality of reference of client request stream, using a metric independent of the cache management policy. Then, since most policies can benefit from the enhanced locality, the expected number of cache misses will decrease.

3.3 A Measure of Locality

It is very important to define a cache-independent metric for locality, so that it can be used in the clustering cost formula. A parametrized metric for locality is the working set size ($WSS(M)$) [Den68]. WSS is the expected cardinality of the set $R_t^{(M)}$ of M consecutive frame requests starting

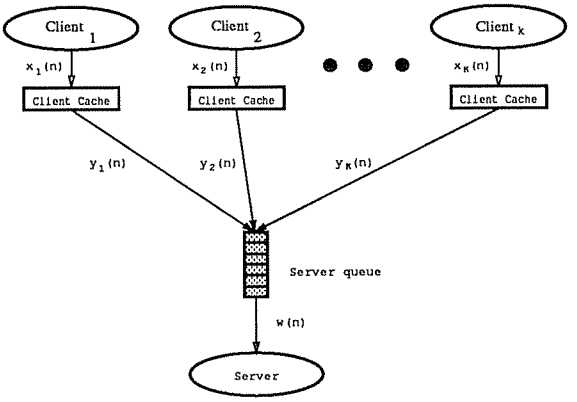


Figure 4: The revised system model

at time t (also used in [YW73]). That is: take these M frame 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. Therefore, in our case WSS (denoted as $K_t^{(M)}$) is:

$$K_t^{(M)} = E(R_t^{(M)})$$

Obviously $1 \leq K_t^{(M)} \leq M$. If $M > N_f = ||D_\pi(S)||$ then the upper limit is N_f (= the number of frames the whole object space S maps to). The window parameter M allows to optimize for different cache sizes, because the smaller $K_t^{(M)}$ is, the more effective a cache of size M is. If the cache size is $\geq N_f$, any replacement policy will work, since the whole object base fits into the cache.

The distribution of the $K^{(M)}$ is independent of time t for IID and time invariant Markovian models, due to their ergodicity properties. From this point on we will drop the t subscript when we refer to those models. Consequently, the WSS formula is:

$$K^{(M)} = E(R^{(M)}) \tag{3}$$

3.4 Revised Model Clustering

To take into account multiple clients and caching, we need to revise the simple system model of Section 2.1. Now the new system model contains a number of clients (K). Each client i has a local access stream, denoted by $x_i(n)$. All access streams follow the same access model, and they are statistically independent. A frame cache exists between each client and the server. Cache misses, denoted by $y_i(n)$, form the client requests that reach the server through a queue of a certain policy (FIFO, Priority, etc). Queueing makes the combined global request stream (denoted as $w(i)$) behave as an IID stream. Its probability vector is the sum of the (stationary) probability vectors of the cache misses $y_i(n)$ (Figure 4).

Therefore, optimal clustering for the new model is defined as two separate problems:

1. *Choose a partition scheme D_π that minimizes the working set size $K^{(M)}$ of the client frame request stream ($D_\pi(x_i(n))$), for a given value of the window parameter M.*

2. *Choose a permutation scheme D_s, such that: given a combined global request stream $w(n)$, D_s minimizes the server frame request cost.*

To solve the second problem, just the statistical description of the global IID stream is needed. Then, the methods developed in Section 2.6 can be used to give optimal clustering. The global stream probabilities obviously depend on the individual cache policies and it is very hard to calculate them analytically. This problem is a good candidate for dynamic or adaptive solution, where the global stream is inspected for some period, and the probabilities are estimated from the stream itself.

The rest of this paper is devoted to the first problem. As we will see, partitioning the object space to decrease WSS is not an easy task.

3.5 WSS of the IID Model

It is not hard to come up with a closed form expression of $K^{(M)}$ for the IID case. Appendix B.1 has all the steps of the derivation procedure:

$$\begin{aligned} K^{(M)} &= M - \sum_{q=1}^{N_f} (1 - \pi_f(q))^M \\ &= M - \sum_{q=1}^{N_f} (1 - \sum_{\substack{y: \\ D_\pi(y)=q}} \pi_s(y))^M \end{aligned}$$

N_f is the total number of frames, and $\pi_f(q)$ is the total access probability of frame # q.

It can be shown that K is minimized for every value of M *when each frame f contains objects placed using the probability ranking* (see Appendix B.1). This agrees with results in [YW73], where a different methodology was used.

3.6 WSS of the SMC Model

A partitioned Markov state space, appears at the limit like a Markov process (see Appendix A.1). As a result, accesses of the frame space F due to object accesses in S can also be modeled as SMC.

In Appendix B.2 the derivation of $K^{(M)}$ for the SMC model is given. Using an occurrence random variable $W(M,q)$ (= 1, if $q \in \{x_1, ..x_M\}, 0$ otherwise), the WSS is:

$$K^{(M)} = \sum_{q=1}^{N_f} W(M,q)$$

For $M = 1$, K is always 1. For $M = 2$:

$$K^{(2)} = 2 - \sum_{q=1}^{N_f} \pi_f(q) P_f(q,q)$$

For $M \geq 3$ the formula becomes more complicated, involving higher powers of the P_f matrix Using the results of Appendix B.2 minimizing $K^{(2)}$ corresponds to maximizing L_2:

$$\begin{aligned} L_2 &\equiv \sum_{q=1}^{N_f} \pi_f(q) P_f(q,q) \\ &= \sum_{q=1}^{N_f} \sum_{\substack{x: \\ D(x)=q}} \sum_{\substack{y: \\ D(y)=q}} \pi_s(x) P_s(x,y) \end{aligned}$$

or minimizing G_2:

$$\begin{aligned} G_2 &\equiv \sum_{q=1}^{N_f} \pi_f(q)(1 - P_f(q,q)) \\ &= \sum_{q=1}^{N_f} \sum_{\substack{x: \\ D(x)=q}} \sum_{\substack{y: \\ D(y)\neq q}} \pi_s(x) P_s(x,y) \end{aligned}$$

Minimizing WSS for $M = 2$ is a weighted graph partitioning problem[6] where the weight w of an edge $a \rightarrow b$ is:

$$w(a,b) = w(b,a) = \pi_s(a) P_s(a,b) + \pi_s(b) P_s(b,a)$$

Each partition must have up to a maximum number of nodes and L_2 must be maximized. Alternatively, the problem is equivalent to minimizing G_2, the sum of weights of the all edges crossing the partition boundaries. It is interesting to observe that clustering scheme $C_2 = \{[si], [mp], [le]\}$ minimizes the WSS $K^{(2)}$ for the example of Figure 1.

The weighted graph partitioning problem has been studied previously. According to [GJ79], deciding if there is a partition with cost $\leq J$, is an NP-complete problem. The clustering decision corresponds to partitioning the object graph vertices to $V_1, V_2 ... V_m$, with:

- vertex weight $w(v) = 1$,
- edge weight $w(e) = w(a,b)$ of an edge $e : a \rightarrow b$,
- $\sum_{v \in V_i} w(v) \leq L$, and
- $\sum_{e \in E} w(v) \leq J$.

where L is a limit in the capacity of frames, and E is the set of all edges that cross the partition boundaries. J is the upper limit for the partition WSS sought.

Although the problem is NP-complete, there are some good heuristic algorithms to find close-to-optimal solutions fast. Notably, Kernighan's partitioning [KL70] is a good $O(n^{2.4})$ heuristic algorithm. [Bar84] and [BVJ84] propose asymptotically faster but more complicated algorithms. Since edge costs cannot be arbitrary (they come from the vertex stationary and edge transition probabilities), it is possible that special purpose partitioning algorithms can be used. In fact we are planning to investigate some special purpose algorithms in the future.

[6] We have shown in [TN90] that minimizing WSS for $M > 2$ is a *hypergraph partitioning* problem.

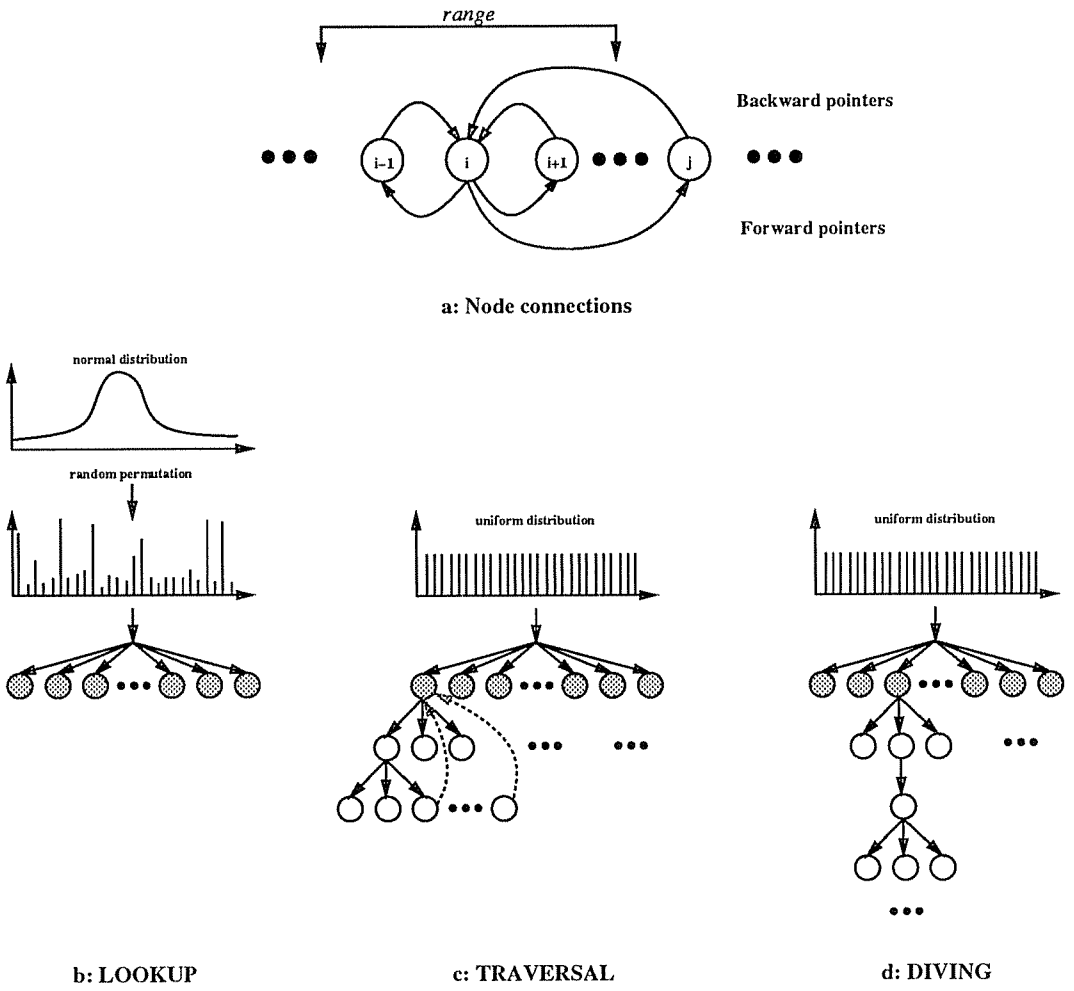


Figure 5: The modified Sun Benchmark

4 Experimental Results

In this section we give some results from our experimental studies on object clustering. The objective of the study was to evaluate various object clustering schemes that have been proposed in the past, as well as some of our own.

4.1 Workload

To pursue any experimental object clustering study, one needs object traces. Since the field of object oriented databases is very young and only a few operational systems exist, we decided to use object references from a benchmark program instead of actual system traces. We chose the Sun Engineering Benchmark [Cat88] because it is widely known, and it provides a common point of reference for other studies as well [DFMV90]. Although it is an artificial example of an object store client, and does not have much of an object oriented flavor (it is written in C), it still produces

some interesting styles of object access patterns. Furthermore, we believe it is a good starting point for deriving more specific benchmarks.

The Sun benchmark builds an object graph of N nodes as follows: Each node is identified by an object identifier (OID) and has 3 children. 90% of the time a child is "close" to the parent (within 1% of N) in terms of OID distance. 10% of the time, a child is selected equally among all N nodes. Each object keeps back-pointers to up to ten of its parents (Figure 5a). Two access styles are defined:

- LOOKUP: K objects are selected among N with uniform probability, and subsequently each one of them is visited.
- TRAVERSAL: K objects are selected like in LOOKUP. Each such object becomes the root of a recursive DFS traversal of the object graph, that goes up to some level D following forward-pointers all times (Figure 5c). Then, another K objects are randomly selected, and another DFS traversal is performed this time using only the back-pointers.

Those two methods fall into the two ends of the spectrum in terms of locality. LOOKUP has very poor locality, since objects are always chosen randomly. TRAVERSAL has a high degree of locality, because of the high predictability of DFS accesses. With 100% probability the first child will be visited after its parent. To cover the rest of the spectrum, we defined two other access styles that are more interesting for the problem of object clustering. First, since no clustering method can perform well under completely random accesses, we modified LOOKUP to perform skewed accesses using normal distribution. The OIDs produced by the normal distribution generator are permuted using a random permutation (Figure 5b), so accidental clustering of accesses is avoided.[7]

Additionally, we introduced a new method called DIVING. DIVING experiences some BFS behavior, descending the object graph up to some depth D. On odd levels, it behaves like BFS, touching first all children of the current node and then selecting the last one of them to follow. On even levels it merely selects one of the children and continues (Figure 5d). As we will see next, DIVING has worse locality than TRAVERSAL, but still better than the modified LOOKUP.

4.2 Performance indices

For this study, we have decided to use the average working set size $\overline{WSS(W)}$ defined in Section 3.3 as our main performance index. Another interesting quantity is the *Working Set Entry Rate*:

$$\overline{DWSS(W)} \equiv \overline{WSS(W,t) - WSS(W-1,t-1)} = \overline{WSS(W)} - \overline{WSS(W-1)}$$

that gives the average rate which a referenced object enters the set of the last W references. At a given time t a cache of size W managed by the working set algorithm will contain $WSS(W,t)$ items of the current working set, and $W - WSS(W,t)$ items from the previous working sets. Consequently, $DWSS$ is an upper bound of the average miss ratio of that cache, since it captures the probability

[7] Without the random permutation hot objects will be "close" to other hot objects, thus automatically achieving a "probability ranking like" clustering.

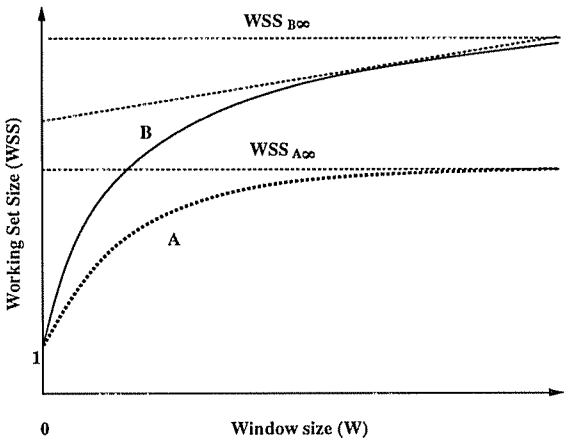


Figure 6: Working set size vs Window size

the currently accessed object x_t does not appear in the working set at time t ($Pr\{x_t \notin WS(W,t)\}$). Note that $\overline{DWSS(W)}$ ranges between 0 and 1, 0 meaning that the next request is *always* in the current working set, and 1 meaning that the next request is *never* in the current working set.

As an alternative to WSS[8] and $DWSS$, we used the average miss ratio of an LRU cluster-cache of size C ($LRU.MR(C)$). In all cases the LRU cache performance was consistent with the $DWSS$, that is, if algorithm A had better $DWSS(W)$ than algorithm B, the same was true for their $LRU.MR(C)$.

WSS is a nondecreasing function of W that asymptotically reaches the working set size W_∞. If the absolute object access probability is skewed (like in our LOOKUP), a small number of hot objects dominates the contents of the working set, and WSS quickly reaches the small W_∞ (like in Figure 6, curve A). When the absolute object access probability is not skewed (as it is the case in TRAVERSAL and DIVE), WSS slowly increases with W to reach a much higher asymptotic value (like in Figure 6, curve B).

In our experiments, the reference stream most clustering schemes produced had significantly better WSS than the original reference stream, simply because a clustering scheme maps a large number of objects into a smaller number of clusters. Between the clustering algorithms WSS varied. Sometimes absolute values of WSS size did not differ much, but the corresponding working set entry rates were very different. For this reason, we will be showing $DWSS(W)$ whenever it is appropriate.

4.3 Clustering Algorithms

The goal of all the algorithms we examined is to partition the object graph stored in the object base, into some number of fixed size non–overlapping clusters.[9] A cluster is identical to a frame,

[8] From this point on we will drop the bar symbol from $\overline{WSS(W)}$, since we will be always referring to the average WSS .

[9] All objects will be assumed to have equal sizes

as it was defined in the previous sections. The object graph (OG) is formed considering objects as nodes and any reference (i.e. pointer) stored in some object a pointing to object b as an edge $a \rightarrow b$. However, not all algorithms use the OG to perform clustering. Instead, they use trace t_0 statistics in the form of a graph, the clustering graph (CG). t_0 assumed to be characteristic of the expected access behavior. The CG contains the same nodes as OG but its edges, edge costs and node costs, are derived from the object trace. To put them into our previous context (see Section 2.1) all algorithms use an access model in the form of a graph. Three types of CGs are used:

- The OG, i.e. the object graph itself. Obviously, no statistical information from the training traces is used in OG.
- The SOG (Statistical Object Graph) is simply the object graph annotated with edge and node weights. The node weight (edge weight) is equal to the number of times the object (edge) appears in the training trace.
- The SMC (Simple Markov Chain) is a stochastic model that has the form of a directed graph. SMC is an estimator of a second order Markov Chain process[10] that could have produced the object trace. The state space of SMC is equal to the set of all objects accessed in t_0, represented as nodes in the SMC graph. Any positive transition probability between two states a, b is represented as an edge $a \rightarrow b$ in the SMC graph, and corresponds to requests of an object b after the object a in the training trace. The node (edge) weights are the the estimated stationary (transition) probabilities of the chain.

The OG does not convey any access information. The last two graphs express the behavior of the client as it is manifested in the training trace, using a much more compact representation than the trace itself. The SOG limits its information in the usage of objects and references, failing to capture access dependencies other than those that exist in the original object graph (for example, SOG will not record a return from a node to its grandparent during a DFS traversal). SMC does not have this type of restriction since it records arbitrary transitions. However, it does not record access paths involving more than 2 objects because of its memoryless characteristics.

Most of the algorithms assign objects to clusters in a *non-backtracking fashion*, that is, after a clustering decision has been made the object is not subject to reclustering. Those algorithms typically traverse the CG in some form of graph traversal and assign objects to clusters. On each step the object is put in to the current cluster, and if the cluster fills up, a new, empty cluster is created.

We tested 6 clustering algorithms. The first two (SMC.PRP and SMC.KERN) algorithms[11] are presented in this paper. The PRP method uses the training trace to compute the probabilities of access for each object in the database, and then uses Probability Ranking Partitioning (defined in Section 2.6) to cluster the objects. The SMC.PRP algorithm has $O(N \log N)$ cost. The SMC.KERN

[10] In second order Markov Chain, the next state only depends on the previous state, so a two-dimensional matrix is needed to record that probability.

[11] We will name the algorithms from the type of clustering graph they use and the style of the graph traversal they perform.

method, described in Section 3.6, uses the training trace to compute an SMC model, and then uses Kernighan's graph partitioning algorithm to find the optimal clustering scheme for that SMC model. SMC.KERN partitions the object graph, so that the expected $WSS(W)$ for $W = 2$ is minimized. KERN is a heuristic partitioning algorithm that achieves only pairwise optimality, i.e., there will be no two nodes belonging to two different partitions that can be exchanged and result in a lower total cost partitioning. SMC.KERN is backtracking algorithm, since it applies repartitioning until no cost improvement is possible. The partioning cost may decrease further if the capacity of each cluster is not fully utilized (i.e. by trading space for locality like in the "simple" example of Figure 1). This can be accomplished by adding dummy (not connected) nodes and repartitioning. Such repartioning as well as optimizing $WSS(W)$ for $W > 2$ was not attempted in this study but it is subject of our future work. The complexity of SMC.KERN is dominated by the complexity of graph partitioning and it is on the average $O(N^{2.4})$[PS82].

The OG.DFS algorithm traverses the object graph in a DFS manner. It minimizes the number of different clusters encountered during a pure DFS traversal. OG.BFS traverses the graph in a BFS manner, grouping siblings together as much as possible. Both algorithms have linear cost $O(N)$.

SOG.WDFS is much like the OG.DFS except that during the DFS traversal siblings are assigned depending on how "hot" their edge is. This algorithm attempts to minimize the number of clusters "probable" DFS traversals will see. Edge heat information can be supplied by the compiler based on static usage information (like in Semantic Clustering [SS90]), or as user hints (like in E [RC88]). SOG.WDFS has linear cost $O(N)$. OG.BFS, OG.DFS and SOG.WDFS have been proposed in [Sta84] for clustering Smalltalk objects.

Finally, the SOG.CACTIS is based upon the clustering algorithm proposed in [HK89]. Quoting from [DK90],

> Clustering starts by placing the most frequently referenced object in the database in an empty block. The system then considers all relationships that go from an object inside the block to an object outside of the block. The object at the end of the most frequently traversed relationship is placed in the block.

To apply this method to the Sun Benchmark, we interpret "relationship" as "edge in the object graph."

4.4 Results

The table of Figure 7 shows the parameters of our study. The object base was created using the specified parameters. We decided to create a small database ($N = 200$) to reduce the memory requirements of the simulation and allow the clustering algorithms to work in reasonable time. Also, with small object base it was much easier to visually inspect the database to verify the correct operation of the algorithms and understand the results. In some cases we tested larger object bases, but no significant changes to the results were observed.

The LOOKUP workload was generated using a normal distribution pseudo–random number generator ($normal(\frac{N}{2}, 0.10N)$) and a random permutation, as explained in the last section. The

OBJECT BASE		
N	Number of objects in the benchmark database	200
range	How close to the parent are the children	1% of N
WORKLOAD		
depth	Depth of the recursive traversals	4
count	How many times top level objects are selected	-
tsize	Size of the trace in terms of object references	5K,10k,20k,40k
type	Type of object accesses	LOOKUP=L, DIVE=D, TRAVERSAL=T
CLUSTERING PARAMETERS		
L	Cluster size	5..30 objects/cluster
PERFORMANCE INDICES		
$WSS(W)$	Average working set size	$W = 2..10$
$LRU.MR(C)$	Miss ratio of the LRU cluster cache	$C = 10..40\%$ N
$NWSS(W)$	Average normalized working set size: $NWSS(W) = \frac{\overline{WSS(W)}}{W}$	$W = 2..20$
$DWSS(W)$	Average working set entry rate: $DWSS(W) = \overline{WSS(W)} - \overline{WSS(W-1)}$	$W = 2..20$

Figure 7: The simulation parameters

TRAVERSAL and DIVING workloads were created using the standard uniform probability pseudo–random number generator to select the top–level objects. Then, the count parameter of the traversal was adjusted to get the desired trace sizes. The depth of the recursion was set to 4 for both workloads, and as a result, TRAVERSAL visited $1 + 3 + 9 + 27 + 81 = 121$ objects per traversal, whereas DIVING visited $1 + 3 + 1 + 3 + 1 = 9$ objects per traversal. We must note that the actual number of references per traversal is slightly larger, because of the way references are generated. For example, in the TRAVERSAL workload, when a DFS traversal from a child returns, the parent is reaccessed to get the reference for the next child.

We generated several random traces for each one of the three workloads. Although we produced them using the same generation parameters, the random number generator was initialized with a different seed in each case. As a result those traces are "statistically similar", thus useful to test clustering on "similar" workloads. The first trace (t_0) was used as a training trace, and the rest (t_1,t_2...) as test traces. In addition, for each workload multiple size traces were produced, so we could observe the effects of using the statistically incomplete information presented in short samples. From each training trace and object graph, each clustering algorithm produced a clustering scheme.

Cluster sizes (L) varied from 5-30 objects per cluster. The choice of L was intended to help us investigate different allocation policies. 5 is very close to the natural object size (the object and its 3 children), but most likely, it is unrealistic since many more typical "small objects" will fit in a (say) 4k page. $L = 30$ models better the small object case allowing approximately 130 objects per

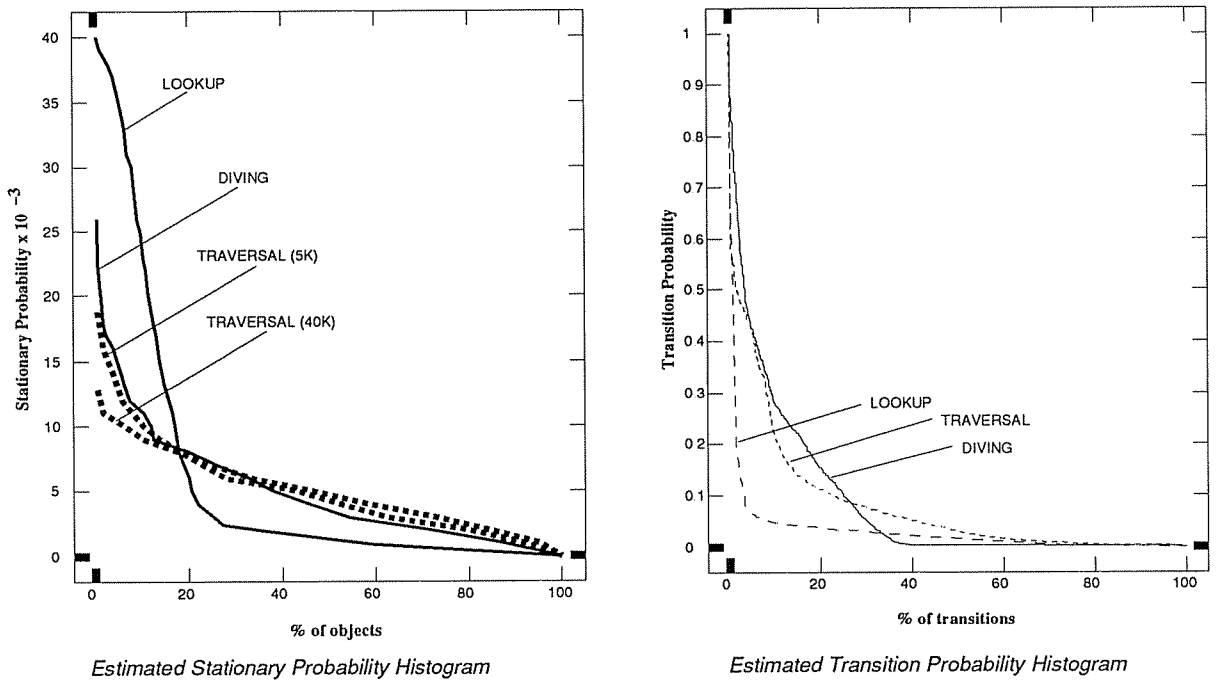


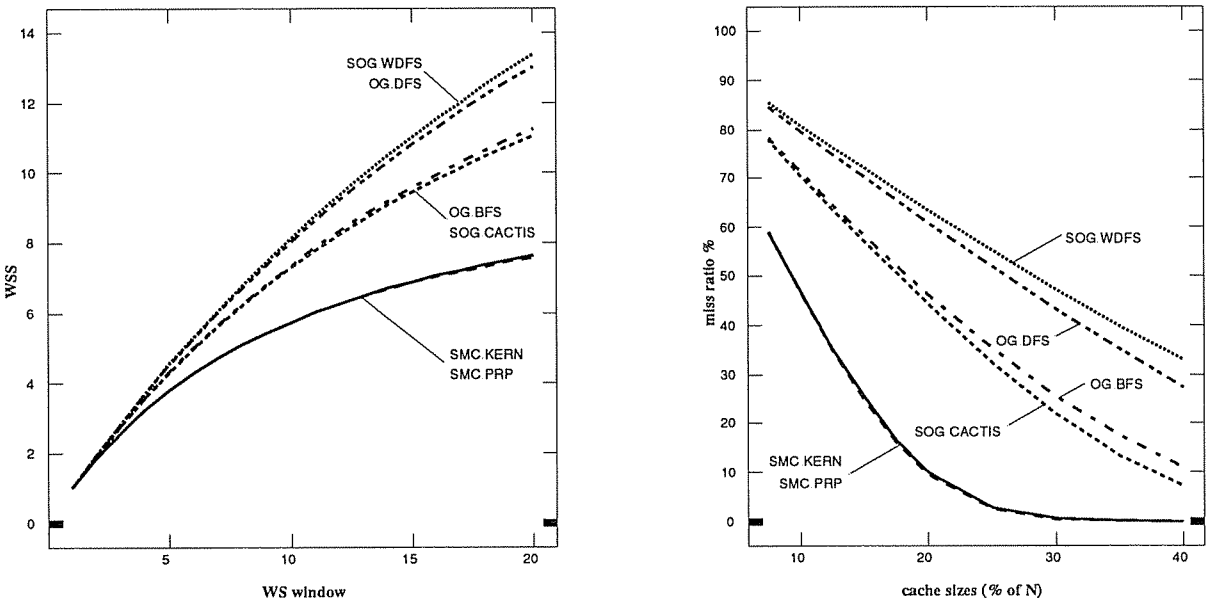
Figure 8: Stationary and Transition Probability Histograms

4k page. In this study with we have assumed all objects have the same size and all pages can the same number of objects. All pages were completely filled with objects, except for the last one of course.

The training trace t_0 was used to derive the necessary access models for the clustering algorithms. Then, each clustering scheme was tested against its training trace t_0 as well as against each one of the testing traces $t_1, t_2 \ldots$.[12] In all cases, the average working set size was calculated as a function of a window of size $W = 1..20$. In some cases, we produced WSS of W up to 100. For selected experiments, the $LRU.MR(C)$ was calculated for cache sizes $10 - 40\%$ of the object space. Finally the WSS of the pure (unclustered) object access stream of each trace was calculated and used as a reference.

After the object traces were generated, we checked their access characteristics used by the clustering graphs (SOG and SMC), as defined in Section 4.3. The first graph of Figure 8 shows the histograms of the estimated stationary probabilities of object access for each one of the three workloads and a variety of trace lengths. LOOKUP produces a relatively small number of very hot objects (approximately 20% of N). In contrast, DIVE and TRAVERSAL created many more hot objects, and their access distribution was less skewed. The next graph shows the histogram of transition probabilities as estimated by the SMC model (Figure 8b). In LOOKUP very few of them had high probability because of the independent accesses. DIVE and TRAVERSAL produced more than 20% of total edges with access probabilities greater than 0.20. DIVE had more edges

[12] When the results from t_1 and t_2 did not agree, we used more traces and averaged the results.



a: Working Set Size (average)

b: LRU cache Miss Ratio (average)

Figure 9: The performance on LOOKUP

with high probability than TRAVERSAL, since 50% of the time it chooses to follow just one of all outgoing edges, whereas TRAVERSAL always follows all outgoing edges. Finally, as all graphs show, the above statistical characteristics of the traces do not change much with respect to the trace sizes, except for the case of TRAVERSAL. The reason is that those characteristics do not directly depend on the trace size, but on the number of traversals performed. In general, the TRAVERSAL workload visits 121 objects per traversal, whereas LOOKUP accesses visit 1 and DIVING 9.

4.5 The LOOKUP workload

The two graphs of Figure 9 and Figure 10 show the WSS and $LRU.MR(C)$ for the LOOKUP workload (40k trace and $L = 5$ objects per cluster). For all performance indices the order of algorithms was:

$$SMC.PRP \approx SMC.KERN > SOG.CACTIS > OG.BFS > OG.DFS \approx SOG.WDFS$$

All algorithms based on the graph structure only performed badly (BFS/DFS). DFS and WDFS performed identically since there are no "preferred edges" to distinguish the two schemes. The Probability Ranking algorithm (SMC.PRP) performed the best. This was expected, since the IID assumption about the trace was valid. The KERN algorithm performed identically to PRP, since it considers the same information as the PRP and tries to do the "right thing" in terms of minimizing the $WSS(W)$. Finally, CACTIS lies in the middle, since it has some of the PRP flavor, taking into account the stationary probabilities.

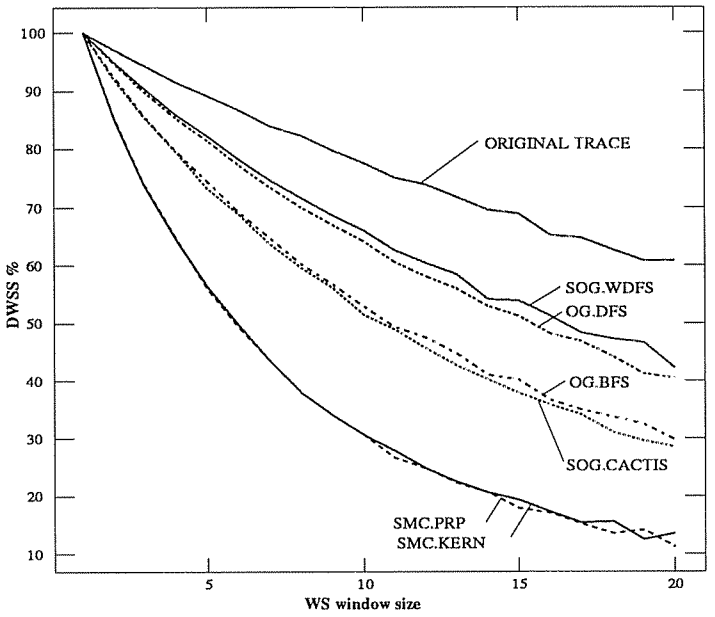


Figure 10: The $DWSS$ on LOOKUP

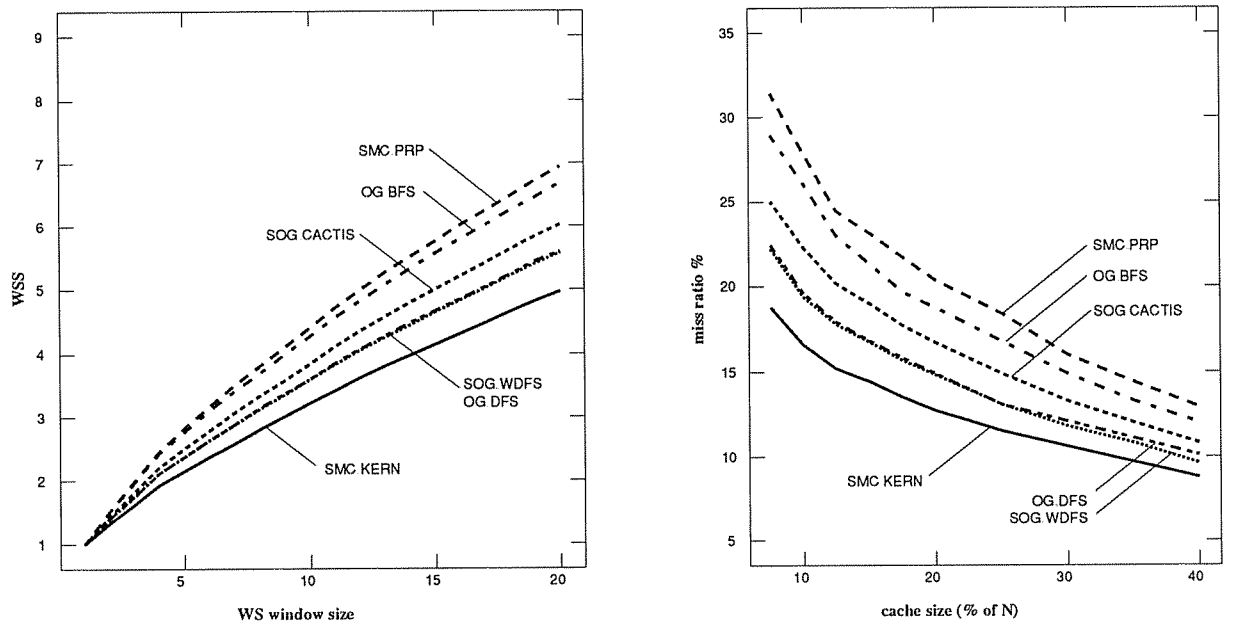
The plot of the WSS in Figure 9 quickly reaches its asymptotic value for PRP and KERN, which is approximately 9 clusters for this workload. This is verified from the $LRU.MR(C)$ plot, where the miss ratio drops almost to zero for a cache size of $C = 35\%$ of the database (10 clusters). The other algorithms do not present this effect, because they do not manage to achieve a reasonably small working set. As it is expected, increasing the cache sizes decreases the miss ratio for all algorithms. Finally the $DWSS$ plot in Figure 10 demonstrates the amount of improvement to the working set entry rates SMC.KERN and SMC.PRP have achieved. On different trace sizes, the order of the algorithms is maintained, but relative differences change. As the trace size decreases, the difference between DFS and WDFS decreases too.

4.6 The TRAVERSAL workload

Figure 11 shows the WSS and $LRU.MR(C)$ for this workload (40k trace and L=5 objects per cluster). The order of algorithms was:

$$SMC.KERN > SOG.WDFS \approx OG.DFS > SOG.CACTIS > OG.BFS \geq SMC.PRP$$

The SMC.PRP (Probability Ranking algorithm) was the worst. Many objects are equally hot and absolute probability is not good clue for clustering in this case. The IID assumption about the trace is obviously not valid, since the probability to access anything but a child of the current node is almost zero. OG.BFS behaved much like SMC.PRP in this case, since objects selected as start of a traversal tend to have higher probabilities than the rest. SOG.CACTIS performed better than OG.BFS, because it has some DFS flavor in grouping the parent and its children together. It



a: Working Set Size (average)

b: LRU cache Miss Ratio (average)

Figure 11: The performance on TRAVERSAL

performed worse than DFS/WDFS because it always clusters objects connected by object graph edges before considering unrelated objects. This is not always the correct thing to do in a depth-first search; in particular, backtracking will result in consecutive references of objects that need not be joined by any edge in the object graph. Both DFS algorithms performed well, since TRAVERSAL is basically a series of DFS traversals that start from random roots (for each root 121 objects are visited in a DFS manner). OG.DFS and SOG.WDFS performed identically because all edges from the parent to the children were equally hot (as in LOOKUP). From all graphs it is evident that for this workload, the SMC.KERN algorithm performed the best. Unlike LOOKUP, the plot of the $WSS(W)$ for TRAVERSAL in Figure 11a did not reach the asymptote for any of the clustering algorithms.[13] Instead it kept rising, with the window size W, but at an approximately fixed rate. On each exhaustive DFS traversal 121 objects are accessed, most of them more than once (this creates the locality). 121 objects is a significant percentage of the object base, and as a result, the WSS cannot settle fast.

This effect becomes more clear in the graph of Figure 12, where $DWSS$ is shown. The improvement that SMC.KERN achieves for $W = 2$ (indicated by a black bullet) is not matched by any other algorithm. It is interesting to compare $DWSS$ of TRAVERSAL (Figure 12) with $DWSS$ of LOOKUP (Figure 10). The randomness of the LOOKUP workload makes it impossible to identify objects accessed in sequence, so even clustering based on the access probability (PRP/KERN) does not help much in terms of the $DWSS$. In the case of TRAVERSAL, such short term prediction is possible, therefore the algorithms that use access statistics are more effective in reducing $DWSS(2)$

[13] As a matter of fact, it did not converge even at $W = 100$.

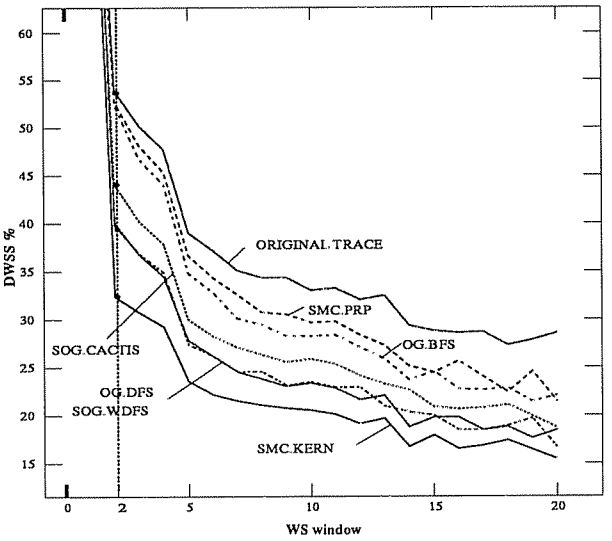


Figure 12: The $DWSS$ on TRAVERSAL

(SMC.KERN to 32% and DFS/WDFS to 40%).

As the window size increases other methods improve too, as an influence of the improved locality of the original trace. The original trace experiences a major drop of its $DWSS$ from 100% to approximately 50% when the window is 4, because at the leaves of a DFS traversal a window of size 4 always contains the parent, and then, half of the references are in the $WSS(4)$.

On the $LRU.MR(C)$ plot, all algorithms start from lower miss ratio than in LOOKUP, because of the better locality of TRAVERSAL. However the miss ratio drops more slowly than in LOOKUP, and never approaches 0, since there is no hope to fit the whole working set in memory. The difference between KERN and PRP ranges between 13% and 4% as the cache size changes from 10%-40% of the database. The relative order of the algorithms is the same as in the WSS plot of Figure 11.

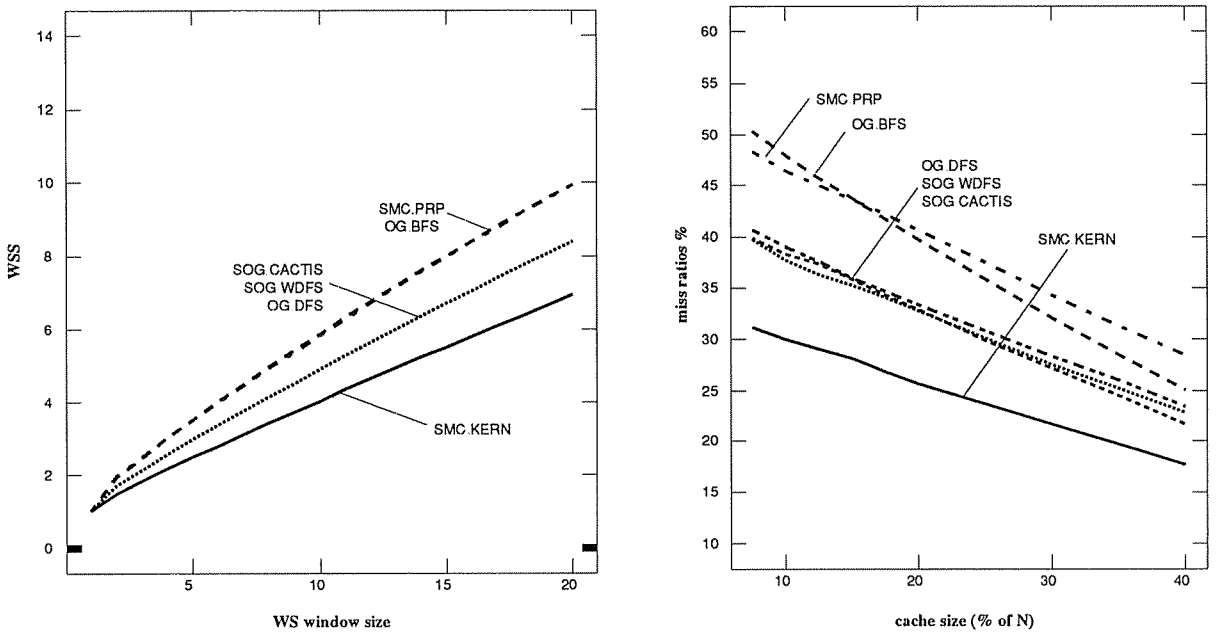
As in LOOKUP, the picture remains the same when going to different trace sizes. The main results is that as the trace size decreases the difference between BFS and PRP decreases too. The reason is that smaller trace sizes cause fewer objects to be selected as start of the traversals. As a result, their access probability is distributed more like the BFS order, i.e., top objects are hotter than their children.

4.7 The DIVING workload

The DIVING workload results are shown in Figure 13. The relative order of the algorithms is:

$$SMC.KERN > SOG.WDFS \approx OG.DFS \approx SOG.CACTIS > OG.BFS \approx SMC.PRP$$

The Probability Ranking algorithm (PRP) and BFS were the worst, for the same reasons as for the TRAVERSAL workload. Unlike TRAVERSAL, DIVING performs many more traversals for the same trace size, because fewer objects are visited per traversal. As a result, the differences



a: Working Set Size (average)

b: LRU cache Miss Ratio

Figure 13: The performance on DIVING

between BFS and PRP are smaller and only show up in small traces (5k). CACTIS did better here because it clusters the parent with its children, and this is right 50% of the times (DIVING touches all children of a node on every odd level of its descent). CACTIS's clustering follows the pattern of DIVING, since the low probabilities of unvisited siblings prevents them from being considered for clustering. Once again, DFS and WDFS have the same performance. 50% of the time they do the right thing, since they cluster a parent node with at least one of its children.

As in TRAVERSAL, the plot of the WSS in Figure 13 does not reach its asymptote. However the $DWSS$ plot shows wider differences in the rates, compared to that of TRAVERSAL (Figure 14). The $DWSS$ of the original trace drops to almost 60% for $W = 3$ again, for the same reason as in TRAVERSAL. A window of that size will always contain the parent of a node which is repeatedly accessed on odd levels of the traversal. It is interesting to observe how much improvement SMC.KERN offers for $W = 2$ and $W = 3$.

In terms of $LRU.MR(C)$, there is constant difference in the performance. SMC.KERN achieves miss rates that are consistently better than the other algorithms: 30% for small caches, versus 40% for the WDFS,CACTIS,DFS and 50% for BFS,PRP. The miss rates drop as the caches get bigger, and the relative difference decreases to 6% and 7% respectively, for large caches. Finally, we have observed that as the trace sizes increase, BFS/PRP difference decreases but the relative order of the algorithms remains exactly the same.

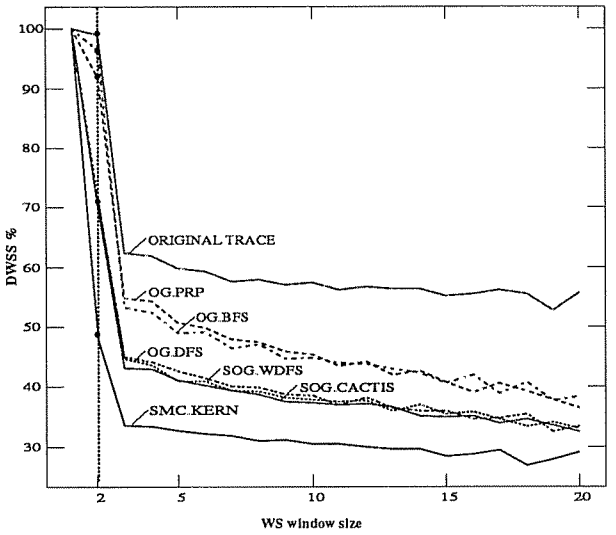


Figure 14: The $DWSS$ on DIVING

4.8 Other simulation parameters

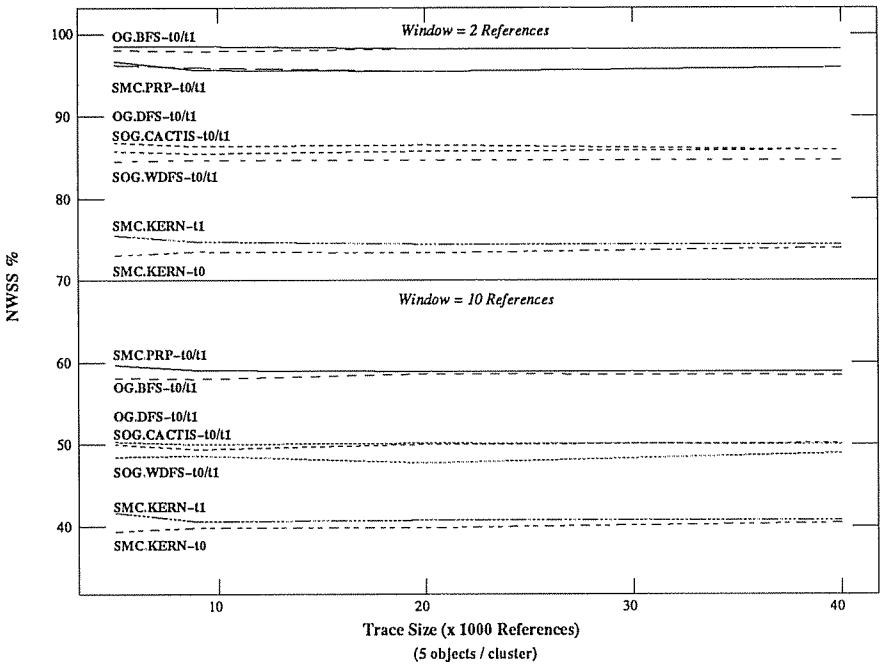
The cluster size $L = 5$ was selected to illustrate better the difference between the algorithms but we ran all the experiments for cluster sizes of 10, 20 and 30 objects.

As we expected, the increased cluster size helped all algorithms to achieve better performance. The graph of Figure 15b shows that SMC.KERN remained the best and its performance improvement over the rest algorithms did not significantly change. For window size 10, SMC.KERN achieved $WSS(10) = NWSS(10) \times 10 = 4$ with cluster size 5, and the next competitor (SOG.CACTIS) needed a cluster size of 15 to match it. For window size 2 (this is what SMC.KERN really optimizes for) no algorithm matched the performance of SMC.KERN for the whole range of cluster sizes.

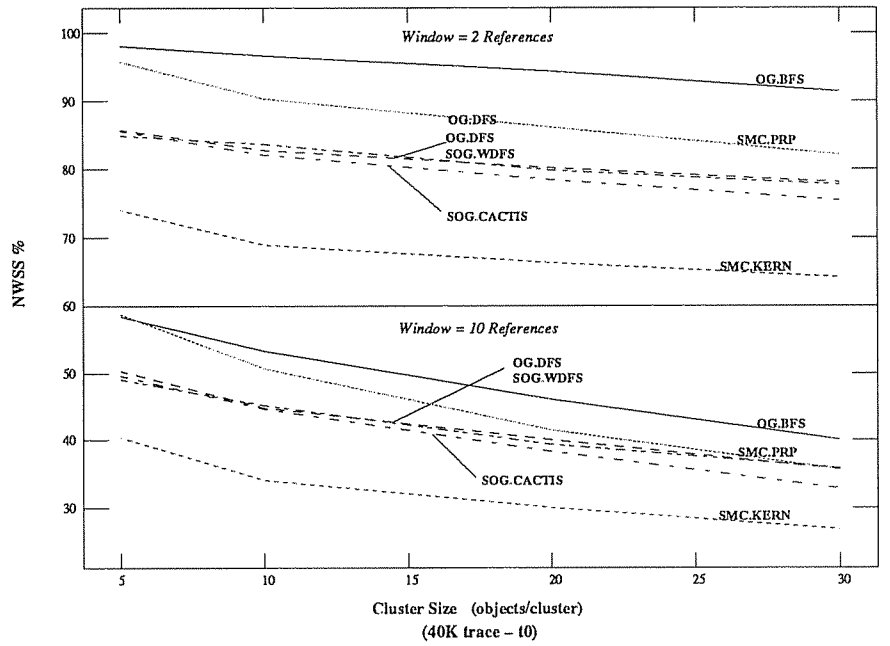
Using traces t_1 and t_2 for testing the clustering based on trace t_0, revealed that most of the algorithms perform the same as with t_0 (Figure 15a). Small trace sizes do not contain "sufficient statistical information" and as a result, clustering based on it is affected. As expected, OG.BFS and OG.DFS were not affected at all, since they use no statistical information.[14] The rest of the methods were affected somewhat, especially on small training traces (as Figure 15b shows). To our surprise, no significant changes have been observed for all traces tried and sizes from 5k-40K references. The algorithm affected the most was SMC.KERN, probably because its performance is heavily based on the SMC model, which is statistically inaccurate on short traces.

To see the effects of changing access patterns on the clustering algorithms, we compared the performance of a trace from the TRAVERSAL workload on a clustering scheme derived from the DIVING workload, with the performance of the same trace on its normal clustering scheme. The performance for the static algorithms (OG class) was unchanged, and the performance of the SOG and SMC class algorithms was affected. Once again, SMC.KERN was affected the most, but it

[14]The small 1% differences, are due to randomness of the short traces.



a: The effect of different traces and trace lengths on DIVING



b: The effect of different cluster sizes on DIVING

Figure 15: The effect of other simulation parameters on DIVING

Workload	Rank
LOOKUP	$SMC.PRP \approx SMC.KERN > SOG.CACTIS > OG.BFS > OG.DFS \approx SOG.WDFS$
TRAVERSAL	$SMC.KERN > SOG.WDFS \approx OG.DFS > SOG.CACTIS > OG.BFS \geq SMC.PRP$
DIVING	$SMC.KERN > SOG.WDFS \approx OG.DFS \approx SOG.CACTIS > OG.BFS \approx SMC.PRP$

Table 1: Summary of the Results

remained better than the others, and still better than the static algorithms.

4.9 Discussion

We have presented the main results of our experimental studies. Table 1 summarizes the relative performace of the clustering algorithms. The three different workloads demonstrate the instability of clustering algorithms not based on statistical information (OG.DFS, OG.BFS). The class of algorithms that use the SOG graph performed well, except for the LOOKUP workload, where they ranked last. The SMC.PRP algorithm worked well only on IID workloads, so it is not suitable for environments were traversals are common. Our algorithm (SMC.KERN) was consistently the best.

So, is SMC.KERN the algorithm of choice for clustering? One can easily see some practical problems with it. First, SMC.KERN requires statistics of not only the object graph usage, but also on requests that may not correspond to traversing an object graph edge. In theory, for N objects, N^2 such possible edges might exist. Second, it is a quadratic algorithm, and thus much more expensive than the others. Finally, the performance improvement observed in terms of locality is not dramatic, especially if we consider large caches.

There are some techniques that could potentially remedy these problems with SMC.KERN. Sampling could be used to decrease the space required for the collected statistics. In some cases, edges and vertices with small probabilities can be discarded, with no significant problems. In fact we have tried this in several cases for all three workloads. The problem is that when the edge probability distribution is skewed, determining a "small probability" can be a tough job. Additionally, we found that some non-backtracking heuristic algorithms with $O(N \log N)$ complexity work well, and they appear to approximate SMC.KERN in terms of performance. Moreover, such partitioning algorithms can be efficiently implemented on parallel architectures (as in [OS90]). Finally, we think that performance improvements of SMC.KERN were substantial even on the TRAVERSAL and DIVING workloads where there exist higher order access dependencies[15] and conflicting access patterns between the forward and backward traversals. But, even if we had that information in the form of a more complex access model, optimizing for larger window sizes would require solving a much harder hypergraph partitioning problem.

[15]The SMC model does not record a dependency of more than two references that the above workloads experience.

5 Conclusions

In the previous sections the advantages of the stochastic approach in clustering have been demonstrated. A realistic system model has been introduced, and the clustering problem has been given a formulation precise enough so that we can rigorously prove optimal clustering for object bases is an NP–complete problem. Furthermore, by reducing the problem to a well–studied weighted graph partitioning problem, clustering strategies can make use of existing heuristic solutions or variants thereof in order to improve the performance of POB systems. Finally, our experimental results have verified that the stochastic approach was indeed correct in terms of improving the locality.

We have pointed out that object clustering has many similarities with data access problems that have been studied in the past. The key difference between those problems and clustering in POBs is in the amount of static, a priori knowledge available about the persistent objects, as well as in the amount of statistical information accumulated over their lifetime. This information, along with the semantics of object oriented systems that restrict the usage of object references to precompiled methods, make stochastic clustering techniques extremely attractive. POBs will be used to store and retrieve large amount of persistent data, possibly residing in distributed servers. Consequently good clustering techniques will be a necessity, and it merits a careful investigation of the underlying problems.

In this paper, we have focussed on what we view as the fundamental underlying problem for clustering algorithms. Much important work remains. Some of the assumptions that we have made need to be further examined. For example, the modeling of client requests remains open, and alternative models (both stochastic and syntactic) should be investigated [TN90]. Answers to questions such as: "how well does the SMC model approximate real-world POB application reference patterns" must await experience with running systems; using the foundation laid by the work presented here, we will be able to make use of these answers as they become available.

Applying the above methodology to a "real system" is not an easy task. Many important questions such as, what to consider as usage data, how often reclustering should be done, how to deal with changes of the object store, and how to cluster in the absence of usage data, need answers. But using the presented framework, one can formally define for the first time, what it is that needs to be optimized, how to optimize it, and how to measure the optimality of solutions.

Acknowledgments

We are thankful to Joshua Chover, Miron Livny, and Anne Condon for their helpful advice and suggestions. We especially want to thank V. Srinivasan for his comments and his careful reading of this paper.

References

[All78] Arnold O. Allen. *Probability, Statistics, and Queueing Theory, with Computer Science Applications.* Academic Press, 1978.

[Bar84] Earl R. Barnes. Partitioning the nodes of a graph. In *Proceedings of the Fifth Quadrennial International Conference on the Theory of Graphs with special emphasis on Algorithms for Computer Science Applications*, pages 57–72, Kalamazoo,Michigan, June 1984.

[BD90] Veronique Benzaken and Claude Delobel. Enhancing performance in a persistent object store: Clustering strategies in O_2. Technical Report 50-90, Altair, August 1990.

[BVJ84] Earl R. Barnes, A. Vannelli, and J.Q.Walker. A new procedure for partitioning the nodes of a graph. Technical Report RC 10561, IBM Thomas J. Watson Research Center, June 1984.

[CAC+84] W. P. Cockshot, M. P. Atkinson, K. J. Chisholm, P. J. Bailey, and R. Morrison. Persistent object managment system. *Software Practice and Experience*, 1984.

[Cat88] R. G. G. Cattell. Object oriented performance measurement. In *Proceedings of the 2nd International Workshop on OODBMS*, pages 364–367, FRG, September 1988.

[Den68] P. J. Denning. The working set model of program behavior. *Comm. ACM*, 11(5):323–333, May 1968.

[DFMV90] David J. DeWitt, Philippe Futtersack, David Maier, and Fernando Velez. A study of three alternative workstation server architectures for object oriented database systems. Technical Report 936, University of Wisconsin, Computer Sciences Dept., May 1990.

[DK90] Pamela Drew and Roger King. The performance and utility of the CACTIS implementation algorithms. In *Proceedings of the 16-th VLDB Conference*, pages 135–147, Brisbane, Australia, 1990.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H.Freeman and Company, 1979.

[GS73] David D. Grossman and Harvey F. Silverman. Placement of records on a secondary storage device to minimize access time. *JACM*, 20(3):429–438, July 1973.

[HK89] Scott E. Hudson and Roger King. Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system. *ACM Transactions on Data Base Systems*, 14(3):291–321, September 1989.

[HZ87] M. F. Hornick and S. B. Zdonick. A shared, segmented memory system for an object-oriented database. *ACM Transactions on Office Information Systems*, 5(1), 1987.

[KL70] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.

[OS90] E. Omiecinski and P. Scheurmann. Parallel algorithm for record clustering. *ACM Transactions on Data Base Systems*, December 1990.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice–Hall Inc, 1982.

[RC88] E. Richardson and M. J. Carey. Persistence in the E language: Issues and implementation. Technical Report 791, University of Wisconsin-Madison, CS Department, March 1988.

[SS90] Karen Shannon and Richard Snodgrass. Semantic clustering. In *Proceedings of the 4-th Int'l Workshop in Persistent Object Systems*, pages 361–374, Martha's Vineyard, MA, September 1990.

[Sta84] James W. Stamos. Static grouping of small objects to enhance performance of a paged virtual memory. *ACM Transactions on Computer Systems*, 2(2):155–180, May 1984.

[TN90] Manolis M. Tsangaris and Jeffrey F. Naughton. Amnesia: a stochastic access model for object stores. Unpublished Manuscript, University of Wisconsin-Madison, August 1990.

[VC90] Paul Vongsathorn and Scott D. Carson. A system for adaptive disk rearrangement. *Software Practice and Experience*, 20(3):225–242, March 1990.

[Won83] C.K. Wong. *Algorithmic Studies in Mass Storage Systems.* Computer Science Press, 1983.

[YW73] P.C. Yue and C.K. Wong. On the optimality of the probability ranking scheme in storage applications. *JACM*, 20(4):624–633, October 1973.

A Appendix

A.1 The stochastic behavior of a partitioned Markov Chain

Here, we will analyze the behavior of a partitioned Markov Chain. Let S be a Markovian state space with stationary probability $\vec{\pi}$ and transition probabilities $P(i,j)$. Let Δ be a a partition of S consisting of non overlapping nonempty subsets of S:

$$S = \bigcup_{C \in \Delta} C \tag{4}$$

The chain S starts at time 0 with some probability vector $\vec{\pi}_0$. Suppose at time n we are given the information that the system is in some state x in the partition C_i. The probability for this event is given by the formula:

$$\pi_n^{(C_i)}(x) \equiv Pr(x_n \in C_i) = \frac{\pi_n(x)}{\sum_{w \in C_i} \pi_n(w)}, \quad x \in C_i \tag{5}$$

where $\vec{\pi}_n$ is derived from the initial probability vector $\vec{\pi}_0$ and the transition probability matrix P:

$$\vec{\pi}_n = P^n \vec{\pi}_0 \tag{6}$$

When n is large (i.e. long time after the start of the chain), S reaches its (unique) stationary distribution[16] and:

$$\vec{\pi}_n \to \vec{\pi}, \text{where } \vec{\pi} = P\vec{\pi}$$

then, the probability vector converges:

$$\pi_n^{(C_i)}(x) \to \pi^{(C_i)}(x) = \frac{\pi(x)}{\sum_{z \in C_i} \pi(z)} \tag{7}$$

The ratio is defined at the limit because we assumed that C_i is non-empty.

The probability that at time $n+1$ the system will be in C_j given that at time n it was in C_i is given by the formula:

$$\begin{aligned}
p_n(C_i, C_j) &\equiv Pr(X_{n+1} \in C_j | X_n \in C_i) \\
&= \frac{Pr((X_{n+1} \in C_j) \cap (X_n \in C_i))}{Pr(X_n \in C_i)} = \sum_{x \in C_i} \frac{Pr((X_{n+1} \in C_j) \cap (X_n = x))}{Pr(X_n \in C_i)} \\
&= \sum_{x \in C_i} \frac{Pr(X_{n+1} \in C_j | X_n = x) Pr(X_n = x)}{Pr(X_n \in C_i)} \\
&= \sum_{x \in C_i} \sum_{y \in C_j} \frac{Pr(X_{n+1} = y | X_n = x) Pr(X_n = x)}{Pr(X_n \in C_i)}
\end{aligned}$$

[16] Since S is an irreducible and recurrent Markov process with no traps or closed subsets.

using (5):

$$p_n(C_i, C_j) = \sum_{y \in C_j} \sum_{x \in C_i} \pi_n^{(C_i)}(x) P_s(x, y)$$

Using (7) $p_n(C_i, C_j)$ converges to $P_\Delta(C_i, C_j)$[17]

$$P_\Delta(C_i, C_j) = \sum_{y \in C_j} \sum_{x \in C_i} \pi^{(C_i)}(x) P_s(x, y) \tag{8}$$

So at the limit, Δ becomes an induced Markov process with transition probabilities $P_\Delta(C_i, C_j)$ given by the above formula. Δ is an irreducible and recurred Markov process with no traps or closed subsets.[18] The stationary probability of that chain is given by the formula:

$$\pi_\Delta(C_i) = \sum_{x \in C_j} \pi_s(x) \tag{9}$$

It can be easily verified that $\vec{\pi}_\Delta$ satisfies the matrix equation: $\vec{\pi}_\Delta = P_\Delta \vec{\pi}_\Delta$. This observation is very important, since it allows us to view a partition of the state space as a Markov chain too. However, great care must be taken on what event constitutes a *state* of the induced process.

[17] To avoid subsequent confusion we will use P_Δ to denote the transition probability matrix for a partition Δ in contrast to P_s for the transition probability matrix of the initial state space S.

[18] If there is a closed set in Δ,then the initial chain contains it too. But S was assumed to have a single closed set only.

B Calculating WSS

B.1 WSS of an IID stream

Using the non-occurrence random variable:

$$U(n,x) = \begin{cases} 0 & \text{if } x \in R^{(n)} \\ 1 & \text{otherwise} \end{cases}$$

Then $K^{(M)}$ is simply:

$$K^{(M)} = M - \sum_{x=1}^{N_f} U(M,x)$$

Because of independence, the probability not to see x in n consecutive references is:

$$U(n,x) = (1 - \pi_f(x))^n$$

so the expected cardinality of n requests will be:

$$K^{(M)} = M - \sum_{x=1}^{N_f} (1 - \pi_f(x))^M \tag{10}$$

It is easy to show that K is minimized for every value of M when each frame f contains objects placed using the probability ranking. Suppose that a partition F that contains the frames:

$$F = \{f_1, f_2, f_3, ...f_i, ..f_j, ...f_{N_f}\}$$

such that $\pi_f(i) \geq \pi_f(j)$, $\forall\ i \leq j$. Now suppose that there are two states $s_1 \in f_i, s_2 \in f_j (i < j)$ such that:

$$\pi_s(s_1) < \pi_s(s_2)$$

Then we can define a new partition F' the same as F with s_1, s_2 exchanged. F''s cardinality will differ from F's only on $U(n,i)$ and $U(n,j)$ terms. Now:

$$\begin{aligned} U(n,i) + U(n,j) &\leq U(n,i)' + U(n,j)', \text{ where:} \\ U(n,i) + U(n,j) &= ((1 - \pi_f(i))^n + ((1 - \pi_f(j))^n \\ U(n,i)' + U(n,j)' &= ((1 - (\pi_f(i) + \gamma))^n + ((1 - (\pi_f(j) - \gamma))^n \end{aligned}$$

with $\gamma = \pi_s(s_2) - \pi_s(s_1)$. This can be derived from the next formula, that can be shown true for any $n > 0$:

$$\begin{aligned} x^n + y^n &\leq (x - z)^n + (y + z)^n \\ z \leq\ x &\leq y \end{aligned}$$

B.2 WSS of an SMC stream

Again an auxiliary random variable will be used, this time measuring the probability of occurrence of a particular state in a sequence of n consecutive states of a SMC model:

$$W(n,x) = \begin{cases} 1 & \text{if } x \in R^{(n)} \\ 0 & \text{otherwise} \end{cases}$$

Then the $K^{(M)}$ is simply the sum of the occurrence variables over all states:

$$K^{(M)} = \sum_{x=1}^{N_f} W(M,x)$$

For $M = 1$, K is always 1. For $M = 2$ there is a simple formula for $K^{(M)}$:

$$\begin{aligned} W(2,x) &= \pi_f(x) + \sum_{y \neq x} \pi_f(y) P_f(y,x) \\ &= \pi_f(x) + \pi_f(x) - \pi_f(x) P_f(x,x) \\ &= 2\pi_f(x) - \pi_f(x) P_f(x,x) \end{aligned}$$

substituting this result in the formula for $K^{(2)}$:

$$\begin{aligned} K^{(2)} &= \sum_{x=1}^{N_f} W(2,x) = \sum_{x=1}^{N_f} (2\pi_f(x) - \pi_f(x) P_f(x,x)) \\ &= \sum_{x=1}^{N_f} (2\pi_f(x)) - \sum_{x=1}^{N_f} (\pi_f(x) P_f(x,x)) = 2 - \sum_{x=1}^{N_f} (\pi_f(x) P_f(x,x)) \end{aligned}$$

and finally:

$$K^{(2)} = 2 - \sum_{x=1}^{N_f} (\pi_f(x) P_f(x,x)) \tag{11}$$

Contents

List of Figures