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

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 a preliminary experimental comparison of this new clustering algorithm with several previously proposed clustering algorithms.

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

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

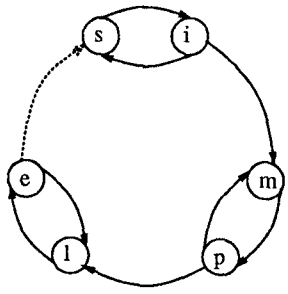


Figure 1: The "simple" example

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 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, assuming an LRU page replacement strategy for the cache, 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 a preliminary 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.

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

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:

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

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), $lim_{n \rightarrow \infty} T_n = \infty$, 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:

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

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.

2.5 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,

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

[3] True for the ID and SMC models.

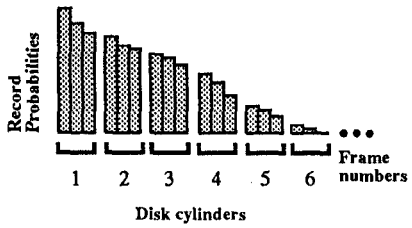


Figure 2: The probability Ranking Scheme

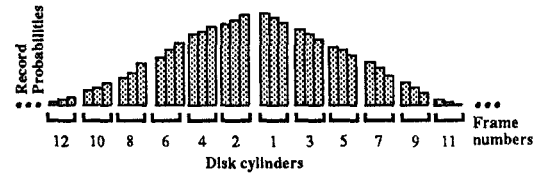


Figure 3: The organ pipe arrangement scheme

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:

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

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 2, where objects with similar temperature[5] are assigned to the same frame as much as possible. The permutation component is the identity ($D_s(f) = f$).

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:

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

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 3:

$$2n \; ...6, 4, 2, 1, 3, 5... \; 2n+1$$

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 %.

[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.

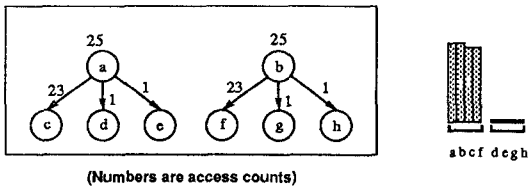


Figure 4: Clustering using probability ranking

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.

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 4, 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 all the 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 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.

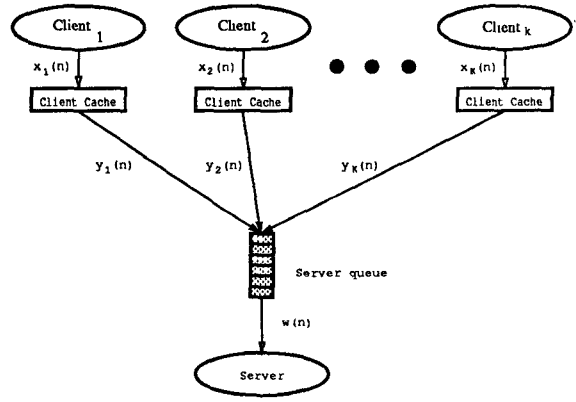


Figure 5: The revised system model

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 5).

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 not trivial 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. [TN91] has all the steps of

the derivation procedure:

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

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* . 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. As a result, accesses of the frame space F due to object accesses in S can also be modeled as SMC.

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 It can be shown that minimizing $K^{(2)}$ corresponds to maximizing L_2:

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

or minimizing G_2:

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

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 $O(n^2)$ 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.

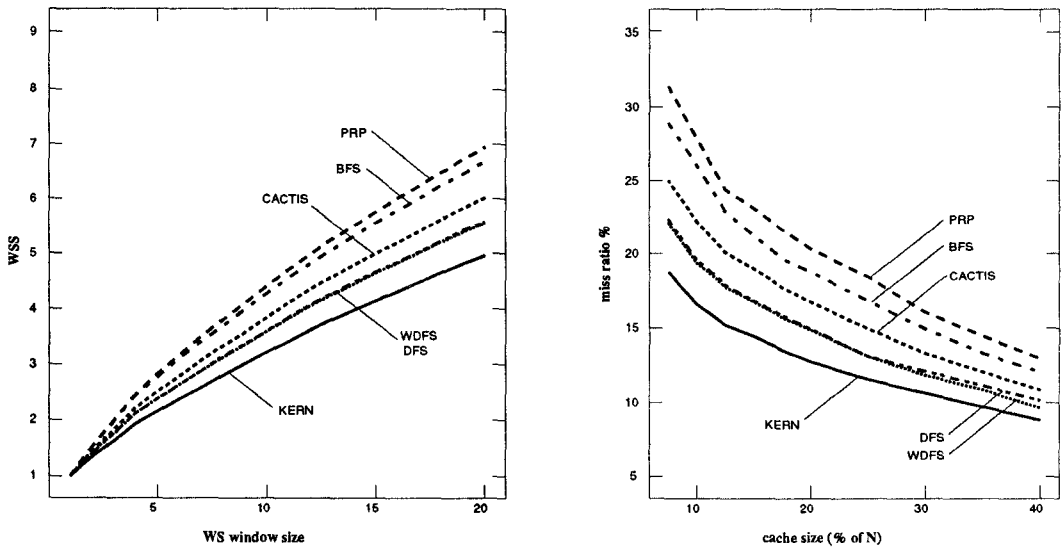
4 Experimental Results

In order to test the stochastic clustering approach, we conducted a series of object clustering experiments. A full description of these experiments and the results can be found in [TN91]. Here, due to space constraints, we present the results of one representative experiment. We present these results as a single interesting data point; obviously, it is foolish to use a single data point to attempt to answer the question "what is the best clustering method[7]."

In this experiment, we used the "traversal" portion of the Sun Engineering Benchmark to generate traces of 5000-40000 object accesses over a database of N=200 objects. (For a description of the Sun Benchmark,

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

[7] In fact, we think that the preceding question doesn't even make sense, since it just raises the question "best for what?".



a: Working Set Size (average)

b: LRU cache Miss Ratio (average)

Figure 6: Clustering performance on TRAVERSAL

see [Cat88].) The traversal portion of the benchmark performs a large number of traversals of the object graph, where each traversal starts from a randomly selected top-level object and performs a four-level deep depth-first search.

We tested 6 clustering algorithms. The first two (PRP and KERN) 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 KERN method, described in Section 3.6, uses the training trace to compute a SMC model, and then uses Kernighan's graph partitioning algorithm to find the optimal clustering scheme for that SMC model.

The remaining clustering algorithms are drawn from previous literature on clustering. BFS and DFS just traverse the object graph in a BFS/DFS manner and assign object to frames in that order. WDFS operates as the DFS algorithm, except that it orders the DFS traversal according to the usage of the object graph edges. BFS, DFS and WDFS were proposed in [Sta84]. CACTIS 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."

The performance metrics we used were the average working set size and the miss ratio of an LRU frame-cache. The graphs of Figure 6 show those metrics for different window sizes and cache sizes (the clustering factor was 5 objects per frame). The BFS algorithm did not perform well, since the workload does not follow a BFS access pattern. Similarly, PRP did not perform well because the trace is not well described by the IID access model (DFS traversal accesses have a strong serial component.) Both WDFS and DFS performed well, since the traversal workload follows a depth-first search access pattern. Our implementation of the CACTIS algorithm 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 moves result in consecutive references of objects that need not be joined by any edge in the object graph. From both graphs it is evident that, for this workload, the stochastic algorithm (KERN) performs the best.

In addition to the traversal workload, we tested the clustering algorithms on a variety of different workloads and clustering parameters (the experiments are fully described in [TN91]). In all cases that we tested, KERN remained the best, while the relative performance of the other algorithms differed significantly for different types of workloads. For example, on a skewed IID workload, the PRP algorithm was as good as KERN.

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.

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.

[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.

[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.

[TN91] Manolis M. Tsangaris and Jeffrey F. Naughton. A stochastic approach for clustering in object stores. Technical Report 1017, University of Wisconsin, Computer Sciences Dept., March 1991.

[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.