Self-Adaptive, On-Line Reclustering of Complex Object Data

WILLIAM J. MCIVER, JR. and ROGER KING

Department of Computer Science, University of Colorado, Boulder, Colorado 80309

email: mciver@cs.colorado.edu and roger@cs.colorado.edu

ABSTRACT

A likely trend in the development of future CAD, CASE and office information systems will be the use of object-oriented database systems to manage their internal data stores. The entities that these applications will retrieve, such as electronic parts and their connections or customer service records, are typically large complex objects composed of many interconnected heterogenous objects, not thousands of tuples. These applications may exhibit widely shifting usage patterns due to their interactive mode of operation. Such a class of applications would demand clustering methods that are appropriate for clustering large complex objects and that can adapt on-line to the shifting usage patterns. While most object-oriented clustering methods allow grouping of heterogenous objects, they are usually static and can only be changed off-line. We present one possible architecture for performing complex object reclustering in an on-line manner that is adaptive to changing usage patterns. Our architecture involves the decomposition of a clustering method into concurrently operating components that each handle one of the fundamental tasks involved in reclustering, namely statistics collection, cluster analysis, and reorganization. We present results of an experiment performed to evaluate its behavior. These results show that the average miss rate for object accesses can be effectively reduced using a combination of rules that we have developed for deciding when cluster analyses and reorganizations should be performed.

1 INTRODUCTION

The benefits of clustering data on disk according to their usage has been recognized for quite some time. Our research examines two central problems germane to clustering in an emerging class of database interactive applications that manage complex data: how to cluster data that has complex structure and are subject to non-traditional types of access methods, and how to adapt physical database organizations, on-line, according to the access patterns being observed.

This research has been supported by NSF awards IRI-8912618 and IRI-9006348, and a General Electric graduate fellowship.

SIGMOD 94- 5/94 Minneapolis, Minnesota, USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

Traditional database systems, or those based on the relational data model, primarily support set-oriented access methods for data items within tuples. In that type of environment, it is generally suitable to cluster tuples of the same type together on disk. However, this approach is not suitable for recent trends in database technology. A new class of database systems and their data models allows the specification and use of complex objects, data items that have complex structure. Whereas objects in the relational data model are flat in structure, restricting tuples to contain one or more atomic attribute values, this new class of data models allows objects to contain non-atomic objects as its attribute values. In addition, objects may share data items with other objects. For database systems that use these types of data models, simply clustering objects of the same type together may not be sufficient since a complex object might be composed of several objects of different types. Furthermore, objects in this new class of database systems might be subjected to additional types of access methods not found in traditional database systems. *Materialization access methods* - the retrieval of all of the components of a complex object - and *navigational access methods* - the recursive retrieval of particular components of a complex object - may be used in conjunction with set-oriented access methods. Each type of access method may not benefit from the same type of clustering. The clustering of databases to simultaneously support these different types of access methods has not been adequately addressed.

An increasing number of interactive applications are making use of object-oriented database systems to manage the data that they use. Examples include CAD and CASE systems. In interactive systems, the group of objects that a user is interested in may be unpredictable, and may change over time. For example, during a CAD session, the user's focus may shift from one design to another, thus, requiring a whole different group of objects to be materialized. This can render the most recent clustering ineffective.

The problem of clustering in an on-line, interruptible and adaptive manner has largely been ignored. A number of the cluster analysis algorithms in the literature are static and, therefore, cannot adapt to changes in usage patterns [1, 2, 19]. Those that are adaptive do not address the issue of clustering complex objects in a way that is adaptive to different types of access methods [3, 7, 22, 24]. None of the adaptive algorithms address the issue of performing reclusterings on-line. Weikum outlined a number of issues related to the execution of on-line reclustering strategies, where clustering activities can be deferred to make on-line reclustering as non-disruptive as possible [23]. He pointed out that increased lock contention and, hence, user transaction disruption could be a major drawback of the introduction of the reorganization phase of an on-line reclustering strategy. The area of garbage collection has addressed the issue

of user disruption through the development of techniques that limit the scope of collections while remaining effective [10,13]. Most (if not all) commercial object-oriented database offerings lack self-adaptive, on-line reclustering capabilities [9, 15, 16, 18]. Some provide features that could be used explicitly by the user to implement on-line reclustering, but they would likely require detailed, application dependent information to implement. Our work was largely directed by an attempt to address each of these issues above.

We were concerned with creating a cluster analysis algorithm that was capable of adapting complex object placement in the face of changing usage patterns by different access methods, namely set-oriented, navigational, and materialization access methods. Toward this end, our cluster analysis algorithm extends upon existing graph-based clustering techniques. Our approach to making on-line reclustering as tractable and non-disruptive as possible draws upon the recommendations of Weikum and generation-based garbage collection techniques. We decompose clustering into its fundamental tasks, making some of them into entities that do not affect lock contention. We introduce a triggering mechanism for determining when it would be useful to perform reclusterings (i.e. cluster analyses and reorganizations). To reduce disruptions due to reorganization, we decided to limit the scope of each reorganization to those objects that have been accessed since the last reorganization. Finally, the overriding philosophy was to develop a framework for clustering complex objects that would be largely independent of object types and that would not require programmatic instruction from database application developers.

The value of our results will lie in the ability to reduce the average miss rate of database accesses below that of static, off-line clustering. While various clustering methods developed over the years have reported reductions in the number of disk page accesses required from 40% to 95% [2,7,19,24], most have been static clustering methods, requiring the database to be taken off line; while the on-line methods did not specifically address the problems of clustering for complex object models.

In summary, the novel aspects of our research include the following research contributions. First, a cluster analysis algorithm has been developed that can adaptively cluster a database to support mixtures of set-oriented, materialization, and navigational retrievals. Second, a system architecture for performing reclustering on-line and in an interruptible fashion has been developed. Third, an experimental evaluation of our algorithm and architecture was performed.

2 SELF-ADAPTIVE CLUSTERING FOR COMPLEX OBJECT DATA

Our clustering algorithm is designed to adapt the placement of complex object components based on clustering metrics that are tied to information about structural relationships between component objects and to the methods used to access them. The structural relationships that we observe are embedded object references between objects. The types of access methods that we observe are those that are either navigational or set-oriented in nature. Our algorithm combines some features of two known algorithms for complex object clustering.

To make our research as general as possible, our data model is defined at a level that is above the file system but below the level of the typing system that a complete DBMS might offer. Our data model allows arbitrarily complex physical schemas to be defined, from atomic objects to recursively defined complex objects to sets of recursively defined complex objects. This allows us to define physical schemas that parallel the possible physical mappings of schemas from higher level typing systems.

The metrics that we use to guide our cluster analyses are based on the properties of our physical data model and the operations defined for it. Our metrics attempt to capture two fundamental types of database events, *references* and *co-references*. A reference simply represents an individual object access (i.e. a read, write or scan in our data model). A co-reference represents an access to one object followed by an access to a second object that it is structurally attached to. Our metric for references is called *heat* [5]. Its purpose is to express the relevance or popularity of individual objects over the course of a series of user transactions. The heat of an object is the frequency with which it has been accessed. This can be measured as an absolute count or as count per unit time. Our metric for co-references is called *tension*. Its purpose is to express the likelihood that certain pairs of objects will be accessed together over the course of a series of user transactions. The likelihood that a pair of objects will be accessed together is what will determine whether or not they should be stored together on disk.

As an example, a sequence of operations on the physical schema in Figure 1 and the statistics they would yield are shown in Table 1.[1] In the first operation of our example, a selection is performed over the set Products, binding any products whose first attribute is equal to 555-1212[2] to variable L1. In the second operation the second attribute of the object bound to variable L1 is retrieved and bound to L2 The third operation updates the second attribute of object C25 with the data "bill paid".

Our algorithm combines the application of usage statistics in the Cactis algorithm [7] with the selective use of both depth-first and breadth-first traversals from the DAG clustering algorithms [2]. The Cactis clustering algorithm chooses the most frequently referenced object in the database; then, the objects adjacent to it in the graph are clustered with it on the same disk page in a breadth-first manner in decreasing order of the frequency with which they were co-referenced with the first object. This process continues until the

---

1.To give the reader a better understanding of the applicability of our metrics, we depict objects in this example using high-level symbolic names which may give the impression that we are using a typed data definition scheme in our work; we are not. In our system each name would actually be some uninteresting object identifier.

2.The variable *prod* in the selection is a free variable which is bound during a selection in turn to each member in the set. Expressions of the form <var>.<attribute>...<attrribute> are path expressions which, depending on the type of operation, return or point to the value of the last attribute in the series. For example, the expression *prod.1* means to return the first attribute of prod in the contexts of SELECT or READ statements; L2.2.2 in the context of a WRITE statement means to point to the second attribute of the object that the second attribute of L2 points to.

current disk page is filled, at which time the most frequently referenced object of the remaining unclustered objects is chosen and the process above is repeated. However, one drawback of the Cactis clustering algorithm noted in [4] is that it cannot cluster objects in a way that is ideal for set-oriented queries. The DAG clustering algorithms group objects based on the traversal order of a graph that represents the database of interest. Breath-first traversals were found empirically to produce clusterings suitable for set-oriented queries and depth-first traversals produce clusterings suitable for materialization and navigational queries. However, usage statistics are not considered in their algorithms.

The cluster analysis algorithm we have developed combines the use of heat and tension with dynamic selection of traversal types. The high-level intuition here is that we utilize depth-first traversals for parts of a database where navigational accesses are most prevalent, and breadth-first traversals where set-oriented accesses are most prevalent. Given a graph representation of a database the clustering metrics heat and tension are treated as weights on the nodes and edges of that graph respectively. The type of traversal performed from any node is selected dynamically. The type is dependent upon the type of references measured most often on the object that a node represents. For this purpose the heat metric is specialized into two types, navigational-heat and set-heat. A depth-first traversal is selected at nodes where the navigation-heat is greater than the set-heat. Otherwise, a breadth-first traversal is selected. In the case that navigational-heat and set-heat are found to be equal, we currently opt, arbitrarily, for a depth-first traversal. It is not clear at this stage in our research what harm, if any, this might do. Regardless of the type of traversal selected, the edges are selected in descending order of their tension counts so as to place together the pairs of objects that have been most frequently co-referenced. An ordinary breadth-first or depth-first traversal on a graph implemented with adjacency lists requires running time $O(v + e)$, where v is the number of vertices and e is the number of edges in the graph [8]. Since our algorithm requires edges to be sorted (i.e. by tension) when traversing from any node, the running time for it is at most $O(v + e \log e)$. This algorithm is given below.

ADAPTIVE CLUSTER ANALYSIS ALGORITHM FOR COMPLEX OBJECTS

Input: G(V,E), a directed graph containing all statistics collected by the Statistics Collector. Output: CLUSTER-SEQUENCE, an ordered set of object identifiers.

```
function Cluster_Analysis(G(V,E)) returns CLUSTER-
SEQUENCE is
    -- create a root vertex, 0, for G.
    for each vertex, v ∈ V without parents
        create an edge <0,v> in G;
        tension(0,v) = navigational_heat(v) + set_heat(v);
    end for;
    CLUSTER-SEQUENCE = ∅;
    CHILDREN = all vertices adjacent to vertex 0 sorted in
    descending order of their tension with 0;
    while CHILDREN ≠ ∅
        v = front element of CHILDREN; remove v from CHIL-
        DREN;
        call sub_Cluster_Analysis(G(V,E),v,CLUSTER-
        SEQUENCE);
    end while;
    return CLUSTER-SEQUENCE;
end Cluster_Analysis;

procedure sub_Cluster_Analysis(G(V,E),v,
                               CLUSTER-SEQUENCE) is
    -- determine what type of access pattern is prevalent from this
    vertex.
    CHILDREN = all vertices adjacent to vertex v sorted in
    descending order of their tension with v;
    if navigational_heat(v) ≥ set_heat(v) then
        while CHILDREN ≠ ∅
            v' = front element of CHILDREN; remove v' from
            CHILDREN;
            if v' ∉ CLUSTER-SEQUENCE then
                append v' to CLUSTER-SEQUENCE;
                call sub_Cluster_Analysis(G(V,E),v',
                                          CLUSTER-SEQUENCE);
            end if;
        end while;
    else -- perform breadth-first traversal from v.
        for each vertex, v', v' ∈ CHILDREN
            if v' ∉ CLUSTER-SEQUENCE then
                append v' to CLUSTER-SEQUENCE;
            else
                remove v' from CHILDREN;
            end if;
        end for;
        while CHILDREN ≠ ∅
            v'' = front element of CHILDREN; remove v''
            from CHILDREN;
            call sub_Cluster_Analysis(G(V,E),v'',CLUSTER-
            SEQUENCE);
        end while;
    end if;
end sub_Cluster_Analysis;
```

3 A GENERAL APPROACH TO ON-LINE RECLUSTERING

The approach we propose to make self-adaptive, on-line reclustering more tractable is to decompose each clustering method into three concurrently operating components: Statistics Collector, Cluster Analyzer, and Reorganizer. In our framework, the determination of where objects should be placed (cluster analysis) and the actual placement of those objects (reorganization) are performed as separate activities. The Statistics Collector is a multitasking process that runs asynchronously with respect to the clustering method components and user transactions. It accepts input from user transactions concerning the type and magnitude of operations they performed and updates the collection of usage statistics accordingly. The Cluster Analyzer is a process that runs synchro-

nously with respect to the Statistics Collector through the use of a triggering mechanism. It is responsible for interpreting the current usage statistics and suggesting new page assignments for objects, called a cluster sequence. The Reorganizer is a database transaction that runs synchronously with respect to the Cluster Analyzer through the use of a triggering mechanism. It is responsible for rearranging objects on the physical disk of the database system to match the page assignments suggested by the Cluster Analyzer. These interactions are shown in Figure 2.

There are several advantages to this structure. First, neither the Statistics Collector nor the Cluster Analyzer compete for database locks since they maintain their data outside of the database. Second, a cluster analysis can take place without necessarily requiring a reorganization. For example, when a cluster analysis is complete it may be determined that it is not worthwhile to reorganize the database. Third, it affords us the flexibility of initiating reorganizations at more ideal times after an analysis has been made.

The Statistics Collector examines the statistics reported to it by user transactions to determine if the Cluster Analyzer should be triggered. The statistics are maintained in a directed graph structure in main memory. By keeping the statistics in main memory, we avoid turning read operations into disk writes of statistics. We also avoid contention by transactions for statistics database locks. Heat, tension, and object destruction statistics are maintained by each user transaction and are communicated to the Statistics Collector upon commit. The collector updates the heat and tension counts in the graph accordingly. The graph is accessible by the Cluster Analyzer concurrent with statistics collection through the use of a locking mechanism.

Our Cluster Analyzer triggering mechanism is based on our tension concept, since objects are "attracted" together in our cluster analysis algorithm based primarily on the strength of tension between them. Observing simple tension counts in isolation can be misleading because tension may occur between objects that are already on the same page and, therefore, could not be clustered in a better way. Thus, we define a special occurrence of tension called *external tension*. External tension occurs when a tension arc connects two objects residing on different pages. Its purpose is to express the likelihood that certain pairs of objects would inhibit database performance by being on different pages. Figure 3 depicts tension and external tension between various objects.

Since external tension arcs might be observed between pages that are likely to be in the buffer anyway due to the particular nature of an application's database usage patterns, we monitor the total external tension in relation to total disk accesses measured to that point. If this ratio rises above a specified threshold, the Cluster Analyzer will be triggered. Thus, we use the following rule as our Cluster Analyzer trigger:

(1) if ((total external tension / total disk accesses) > cluster analysis threshold) then

Trigger(Cluster Analyzer);

The Cluster Analyzer examines the graph maintained by the Statistics Collector and determines suitable new clusters for objects in the database, in the form of a sequence of object identifiers, called a *cluster sequence*, in the order that it recommends they be organized on disk. The sequences are also kept in a location in main memory that is accessible to the Reorganizer should it be triggered.

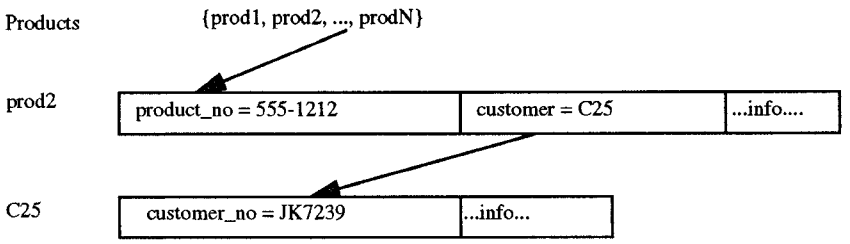

FIGURE 1. Example Physical Schema

TABLE 1. Example Set of Operations and the Statistics They Would Yield

Operations	Results	Statistics Generated

1. L1 := Products SELECT {Prod | Prod.1 = 555-1212}.	L1 = prod2	heat(Products) = 1;
		heat(prod2) = 1;
		tension(Products,prod2) = 1.

2. L2 := READ L1.2.;	L2 = "...info..."	heat(prod2) = 2.

3. WRITE L2.2.2 "bill paid".	C25.2 = "bill paid."	heat(prod2) = 3;
		heat(C25) = 1;
		tension(prod2,C25) = 1.

Once the Cluster Analyzer produces a cluster sequence, it decides whether or not it is beneficial to ask the Reorganizer to enact it. If the cluster sequence is similar to the one used in the previous reorganization, it would probably not make sense to pay the overhead of going through another organization. Our reorganization triggering method measures the dissimilarity between the current cluster sequence and the previous one (if any). We call this measure the *cluster sequence dissimilarity* (CSD). If the CSD (defined below) is found to be above a specified threshold, the Reorganizer will be triggered. We use the following rule as our Reorganizer trigger:

(2) if ((CSD(previous-cluster-sequence,
current-cluster- sequence) >
reorganization trigger threshold) then
Trigger(Reorganizer);

We define a measure of cluster sequence dissimilarity that expresses the difference between the logical placement of two sequences of objects, given their size and position in their sequences. Cluster sequence dissimilarity is expressed as follows:

(3) let a database be represented by
$DB = \{o_1, o_2, ..., o_n\}$ a collection of objects;

(4) let a cluster sequence be represented by CS an ordered set such that $CS \subset DB$;

(5) let a logical assignment of objects in DB to m disk pages be represented by
$P_k = \{p_1, p_2, ..., p_m\}$, a partition of DB
consisting of disk pages each having a capacity of k bytes;

(6) Given $Size : DB \rightarrow N$, a mapping from an object to its size in bytes; and

(7) $Page_k : CS \rightarrow P_k$, a mapping from objects in a cluster sequence to a page assignment of disk pages of capacity k bytes such that:

$$Page_k(o_i) = \left\lceil \frac{\sum_{j=1}^{j<i} Size(o_j)}{k} \right\rceil;$$

(8) then given two cluster sequences CS_{prev} and CS_{curr} the cluster sequence dissimilarity between them is defined as:

$$CSD_k(CS_{prev}, CS_{curr}) = \frac{\left\lceil \sum_{i=1}^{min(|CS_{prev}|, |CS_{curr}|)} Diff_k(o_i, n_i) \right\rceil + D}{max(|CS_{prev}|, |CS_{curr}|)};$$

where $o_i \in CS_{prev}$, $n_i \in CS_{curr}$,
$D = ||CS_{prev}| - |CS_{curr}||$ and

$$Diff_k(o_i, n_i) = \begin{cases} 1, \text{ if } Page_k(o_i) \neq Page_k(n_i) \wedge o_i = n_i \vee o_i \neq n_i \\ 0, \text{ otherwise} \end{cases}$$

Thus, if the cluster sequences match exactly, the cluster sequence dissimilarity between them will be 0. While on the other extreme, a pair of cluster sequences that do not match in a single position, including the empty sequence, will have a cluster sequence dissimilarity of 1.

We used a naive reorganization algorithm for the experiment presented in this paper, whereby the objects in a cluster sequence are moved sequentially to a new set of contiguous pages[1]. We recognize the existence of a more efficient reorganization algorithm by Omiecinski [17]; however, while the object manager we used to implement our database system had many excellent features, its application interface did not provide us with a way of effectively implementing his algorithm. We would hope that the incorporation of Omiecinski's algorithm in a future implementation of our clustering framework would result in better results than are shown here.

We have chosen to make use of logical object identifiers in our database system. This permits us greater flexibility in interrupting a reorganization, it can reduce the number of pages that have to be read during reorganization, and it will permit greater flexibility in developing local algorithms. We can interrupt reorganization at any time as long as the loid table is updated to reflect objects that have been moved up to that time. Without this indirection, we would be required to retrieve and update all the objects that contain references to objects that have been moved or use a special pointer forwarding scheme. Thus, with this indirection we can reduce the number of page accesses required for reorganization. Also, we can precisely focus on limited areas of the database in implementing local algorithms since we are not required to retrieve objects containing physical addresses that required updating when the objects they reference are moved.

4 EXPERIMENTAL FRAMEWORK

It was noted in [4] that controlling the generation of synthetic databases and transactions streams to truly match a natural spectrum of databases and their uses is a very difficult task. We chose a different tack and decided to evaluate our approach in conditions that approximated as closely as possible, given our rudimentary physical database system, real-life database applications that performed complex tasks. At this time, we are only able to show the results of self-adaptive, on-line reclustering for one such application. Based on field experience, it is our feeling that the benchmark presented below is representative of the structure and complexity of tasks of a reasonably wide range of real-life applications.

It was our assumption that one of the major drawbacks of on-line reclustering would be the introduction of increased transaction wait times. It is shown in [21] that the mean waiting time per lock conflict is affected by the distribution of transaction size and that it is proportional to the mean number of conflicts per transaction.

1.This was done using the NEAR_LAST_PHYSICAL hint for the sm_CreateObject function provided by the *Exodus Storage Manager's* application interface.

We, therefore, intentionally focused our initial evaluation on the case of on-line re-clustering for a limited number of competing transactions. This makes us afraid that our results may not be generalizable in terms of the degree of transactions our framework might encounter; however, we feel that a fairly wide class of applications such as CAD and CASE systems might still be well-served by an on-line reclustering framework with limited transaction capacity since they typically employ low numbers of concurrent transactions. Still, future directions for this work must include a study of our framework in the presence of greater degrees of concurrently operating transactions.

The benchmark we chose is called the Trouble Ticket Benchmark [14]. It was chosen because it simulates a real class of interactive on-line systems used by many companies to manage responses to customer complaints. This benchmark is an abstraction of a currently deployed application that utilizes an object-oriented database for its internal data store. It also presented us with a scenario where possibly conflicting clustering needs between transactions would present themselves. A brief description of the benchmark is given below. We direct the reader to [14] for a more detailed description.

In this system, a customer contacts an agent when they experience a problem with a product or service. The agent accesses all of the customer's product information along with any information concerning previous trouble calls the customer has made. The customer's trouble report is then entered into a *"trouble ticket"*. Trouble tickets may contain symptoms, diagnoses, and referrals for a customer's problem. A customer's call may be resolved in two ways. First, the problem may be resolved by the agent during the customer's call, in which case the trouble ticket is closed. In the second case, the agent is not able to resolve the problem during their call and must record one or more diagnoses of the problem and refer (i.e. leave open) the trouble ticket to a handler agent (e.g. a specialist) for resolution. Resolution of a ticket in the latter case may require more than one referral.

These trouble ticket handling processes are simulated in the benchmark by two entities, the *Opener* and *Closer*. The Opener is a repeating transaction that randomly selects a product, retrieves all of the existing information about that product and determines through a probabilistic procedure whether the trouble ticket can be immediately closed, or if it must be left open for further processing. The existing information would be in the form of other trouble tickets and their symptoms and diagnoses. In the case that a trouble ticket remains open, one or more symptoms and diagnoses are added to the trouble ticket and it is placed in a database object called the *open list*.

The Closer is a repeating transaction that processes those tickets that have been left open (i.e. are in the open list), eventually closing them. It determines through a probabilistic procedure whether a ticket should be closed or whether is should be subjected to further diagnoses and referrals. Any trouble ticket that the Closer does not close after removal from the open list is placed in a database object called a referral list. It chooses between the open list and referral list in selecting the next trouble ticket to handle and retrieves all of the existing information contained in the ticket. A ticket is handled by the Closer a finite number of times. This bound is called the handler threshold. Each of the key events in this benchmark, opening, referral, and closing of trouble tickets, are recorded by attaching a special trouble ticket event record to the ticket.

The physical database schema we used to represent the information stored in this trouble ticket process is given in Figure 4. In this diagram, the arcs indicate object references, the blocks represent records, and the braces indicate sets.

The initialization transaction handles the creation of the initial database for the benchmark. Customers are assigned at least one product; 90% are assigned one product; and the remaining 10% of customers are assigned between 2 and 6 products from a uniform distribution. The initialization transaction may also create an initial set of trouble tickets for randomly selected customers, or the benchmark can be run such that the Opener transaction is started ahead of the Closer transaction in order to create enough trouble tickets to keep the Closer transaction busy.

For this experiment, we wanted to evaluate the effectiveness of our cluster analysis algorithm in reducing average disk I/O for our benchmark application. We also wanted to evaluate the impact and effectiveness of our triggering methods for cluster analysis and reorganization and identify possible heuristics for setting trigger threshold values. Finally, we wanted to observe the overhead costs of our clustering framework for the both the benchmark application and clustering system software.

We implemented a database system conforming to our physical data model using the Exodus Storage Manager. The database system includes a lock manager that is external to Exodus. The lock manager permitted us to perform locking at the level of logical object identifiers. Our database system utilizes this lock manager to implement the two-phase locking protocol. Using these components, we then implemented a Statistics Collector, Cluster Analyzer and Reorganizer.

We randomly generated a database of objects conforming to the Trouble Ticket Benchmark schema [14]. As each customer object was created, it was statically clustered in close proximity to its products and other component objects. The resulting database had: 1000 customers, 3795 raw objects (representing customers, products, trouble tickets, etc.), 326 disk pages occupied by the initial database.

We used one Opener and one Closer process. The Opener was started 5 minutes in advance of the Closer to allow the open list to fill up with enough trouble tickets to keep the Closer busy throughout its operation. Both processes were allowed to operate for one half hour per run. The inter-transaction delay for each process was selected from a Poisson random distribution ($\lambda = 10$) in order to induce Poisson processes [20]. The storage manager was started using 104 buffer pages (its minimum). Buffer and disk pages were 4K bytes in capacity. Each run was performed on the same initial database.

Our experiment was organized in three stages in an effort to identify beneficial combinations of cluster analysis trigger (CAT) and reorganization trigger (RT) threshold settings. To present our system with a range of different database usage patterns, the procedures for each stage described below were performed for each of three workloads that correspond to a particular random distribution

used by the Opener to select products. The random distributions were: uniform, normal, and Zipfian[12,25].

In Stage 1, the Opener and Closer were run with a cluster analysis trigger threshold of 0 and an infinite reorganization trigger threshold. This stage had the effect of running the benchmark with no reorganizations (i.e. no on-line reclustering). This stage was intended to serve as a control and to collect statistics regarding the distribution of cluster analysis trigger values. These statistics were then used to set cluster analysis trigger values in Stage 2 of the experiment.

In Stage 2, four runs were performed with the reorganization trigger threshold set to 0 and the cluster analysis trigger threshold set variously to 0 and three other values within the distribution of cluster analysis trigger values observed in Stage 1, namely the mean, -1 standard deviation from the mean, and +1 standard deviation from the mean. This stage had the effect of performing reorganizations after each cluster analysis. This stage was intended to serve as a control for the conservative use of reorganization (i.e. with non-zero thresholds) in Stage 3 and to collect statistics regarding the distribution of reorganization trigger values. These statistics were then used to set reorganization trigger values in Stage 3 of the experiment.

In Stage 3, three runs were performed for each cluster analysis trigger threshold setting in stage two. In each of these runs the reorganization trigger (i.e. cluster sequence dissimilarity) threshold was set to one value within the distribution of reorganization trigger values observed in Stage 2, namely the mean, -1 standard deviation from the mean, and +0.5 standard deviation[1] from the mean.

5 RESULTS AND ANALYSIS

In this section, we present the results of several categories of measurements collected during our experiment. At the highest level, these results show that both of our triggering rules can be used successfully in concert to control our on-line reclustering framework in reducing average disk I/O for user transactions. It will be seen that this is not without a price, however. On-line reclustering is shown to introduce increased user transaction delay due to contention for database resources by the Reorganizer.

It was our goal to show that the cluster analysis and reorganization triggering mechanisms could be used to guide our reclustering framework in reducing the average disk I/O costs over the extent of a benchmark run below that of the statically clustered control runs (i.e. Stage 1 runs). The measure that we use to express this cost is the *miss rate*, or:

(9) *miss rate = buffer page faults / object accesses* [6].

1.In all cases $+\sigma$ would have been out of the range of the distribution of observed cluster sequence dissimilarities.

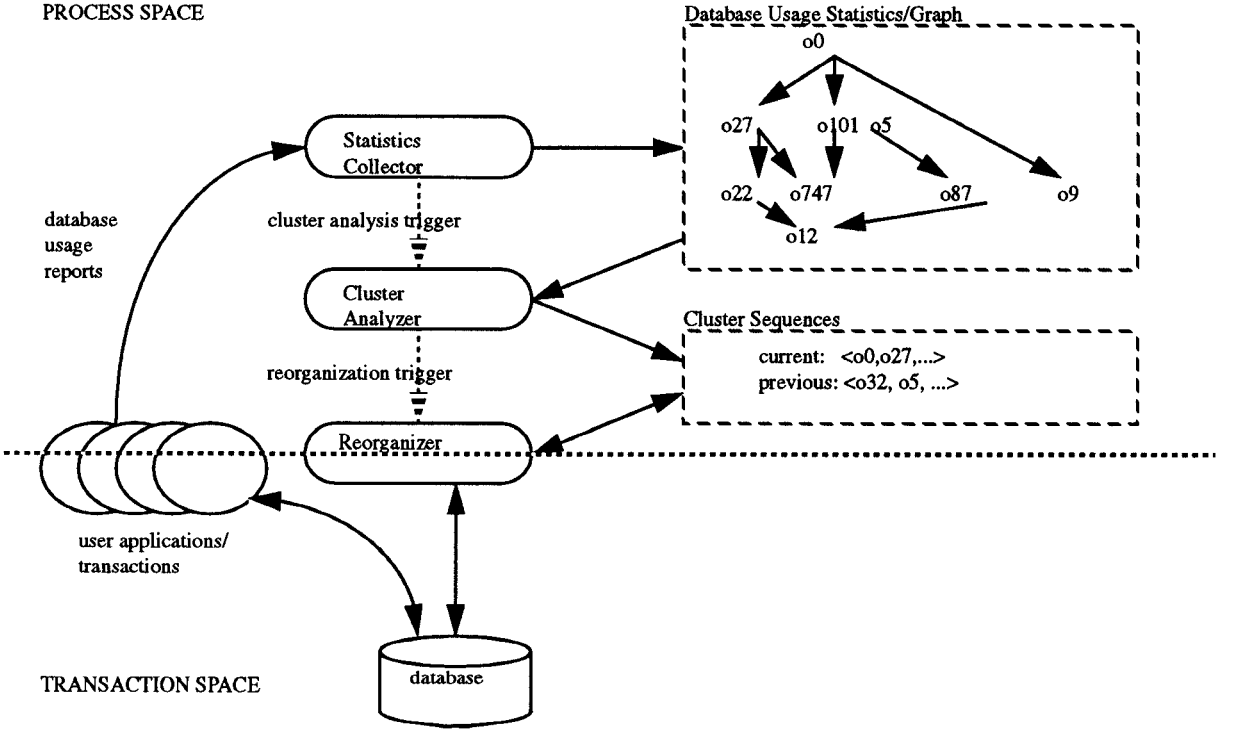

FIGURE 2. Clustering Framework

We compared static clustering against our method was to compare the average miss rate over the life of each benchmark process, where the average miss rate over the entire set of transactions in a run is:

(10) *average miss rate* = Σ *buffer page faults* / Σ *object accesses* [11].

Due to space constraints, we will only show the results for the runs conducted using a normal workload. Figures 5 and 6 show a comparison between the average miss rates over time where self-adaptive, on-line reclustering was performed and where it was not. The comparison is shown for both the Opener and Closer. The results for on-line reclustering are those of the optimal run. The dashed lines in the graphs represent the runs where no on-line reclustering was performed (i.e. Stage 1 runs) where CAT = 0 and RT = ∞; the solid lines represent the results of on-line adaptive clustering (i.e. Stage 2 and Stage 3 runs) where the CAT and RT values are indicated by the caption; and the vertical lines in each graph represent the conclusion of a reorganization, with the horizontal line at the top representing the duration of the reorganization. Note that the duration of the reorganizations shown includes the time that the Reorganizer must wait for user transaction to complete in order to obtain necessary locks.

The most conservative combinations of CAT and RT thresholds were the most effective in reducing the average miss rate. The worst average miss rates were observed when either or both the CAT and RT were set to 0. These results seem to indicate that a policy of liberal reorganization or reorganization that may be triggered by liberal cluster analysis is not helpful in reducing the average miss rate. Rather, more conservative cluster analysis trigger and reorganization trigger thresholds seem beneficial, especially when the access patterns are more localized (e.g. the normal workload).

Another trend that was seen in the results is that reductions in the average miss rate from 7% to 22% for a given CAT threshold were observed as the RT threshold was increased. This indicates that the rationale for using cluster sequence dissimilarity as a criteria for reclustering was generally correct. For those runs where reorganization was recklessly applied after a cluster analysis the data show that the average miss rate was above that of the control runs.

Reductions in the average miss rate were observed only for the Closer; with the exception of one run. A reduced miss rate was expected for those processes that visit an area of the database after it has been reclustered. This effect was reflected in the miss rate results for the benchmark. By the nature of the benchmark, the Closer process had a greater locality of reference than the Opener. Locality of reference for the Opener is dictated by the random distribution used to selected products (i.e. its workload). For the Closer, locality of reference is determined by the trouble tickets it must process in both the open and referral lists. The probability that a process will access the same part of the database (i.e. same trouble tickets) more than once is greater for the Closer since it may leave each ticket open for continued processing by placing them on the referral list and then subsequently access them. This feature in the Trouble Ticket benchmark explains why reductions

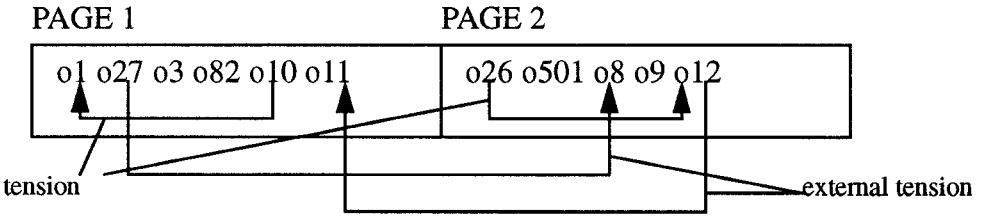

FIGURE 3. Tension versus External Tension

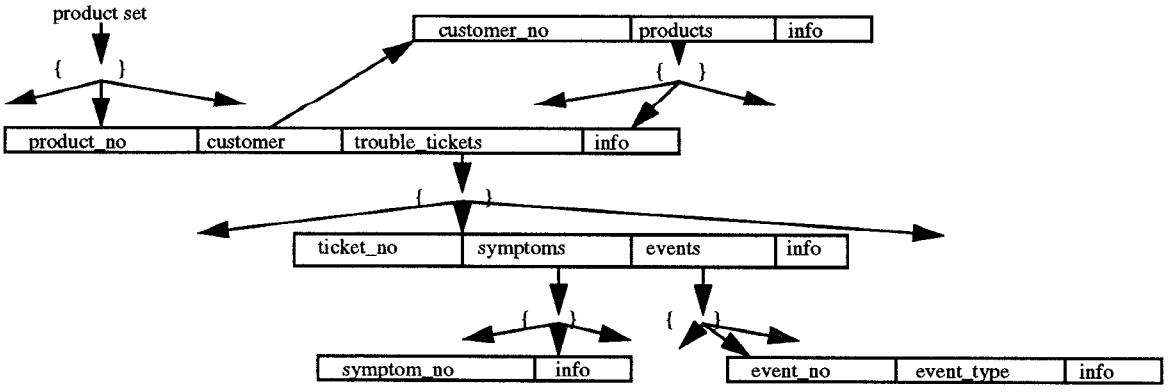

FIGURE 4. Physical Schema for the Trouble Ticket Benchmark

in the average miss rate for only the Closer were observed for that particular workload.

One of the apparent and expected drawbacks of on-line reclustering was that it was observed to reduce the total transaction throughput for both the Opener and Closer. In the Stage 1 runs, the Opener and Closer completed 104 and 110 transactions respectively; while in the optimal Stage 3 run, the Opener and Closer completed 90 and 96 transactions respectively.

Several relationships between reorganization and user transaction wait times are apparent. The average transaction wait time seems to be sensitive to: the number of reorganizations performed during a run, the average cluster sequence length the Reorganizer had to process during a run, and the average duration of reorganizations during a run (the cpu overhead data that is shown later probably also factors somewhat into the delay). Figure 7 shows the relationship between transaction wait times and numbers of reorganizations. There was seen to be a general upward trend in the wait times as much as 300% as the number of reorganizations increased to a certain point. Beyond that point a general downward trend was observed to within 150% of the average wait times observed when no on-line reclustering was performed. Other critical relationships appear to be those between average user transaction wait time and average cluster sequence length (Figure 8) and reorganization duration (Figure 9). In these graphs we can see that the average wait time approaches a peek as the cluster sequence length and the reorganization duration approaches a "critical point". Beyond this "knee" the average wait time begins to taper off.

These last two relationships can be explained by examining two other relationships. The one between the number of reorganizations and the average duration of those reorganizations, and between the number of reorganizations and the average cluster sequence length of those reorganizations. As the number of reorganizations increases during a run, the time interval between them is shortened. As this interval decreases, the statistics that are used to generate a cluster sequence generally cover a smaller number of objects; hence, the Reorganizer is responsible for moving fewer objects. As the duration and quantity of work that the Reorganizer must perform decreases, user transactions do not need to wait as long to gain access to the necessary database locks. Figures 10 and 11 support this contention by showing that reorganization duration and cluster sequence length decrease as the number of reorganizations during a run increases.

Thus, along with the known properties of transaction wait times due to the performance nature of strict two phased locking, increased transaction wait time was also seen to be due to a combination of factors that are introduced by our on-line reclustering framework. First, our use of threshold triggering mechanisms to invoke cluster analysis and reorganization are directly related to the length of the cluster sequences produced. Since we currently re-initialize our statistics collection only after a reorganization, if they are infrequent then the cluster sequence lengths are likely to be large due to the amount database usage statistics that would be collected between reorganizations. Second, the cluster sequence length dictates how long the Reorganizer's transaction will be, or the number of locks it must request, thereby affecting the mean wait time of transactions in the system. Also, the cluster sequence length would also be affected by the average number of unique objects that transactions access since it results in larger numbers of objects being represented by the graph from which the cluster sequences are derived. Finally, since it was seen that reductions in the average miss ratio are not likely with frequent reorganizations, relaxation of the reorganization triggering threshold is probably not a suitable option for reducing transaction wait times.

We were concerned that statistics collection by user transactions would require an inordinate amount of cpu overhead and memory. The cpu overhead required to maintain those statistics during transactions by user transactions (i.e. before they are communicated to the Statistics Collector) was shown to be not insignificant. The average CPU time per object access was seen to increase over the runs where on-line reclustering was not performed 7%, 28% and 36% under the Zipfian, uniform and normal workloads respectively. Clearly, these results indicate that more efficient techniques for user transaction statistics collection must be developed. In our implementation, the size of statistics reports maintained by user transactions is largely independent of the size of the database in question. Rather, it is more a function of the number of unique objects that are accessed by the database since only one unique record is maintained per object for heat reports and per pair of objects for tension reports in our implementation. This policy, however, means that a search is required for an object's heat or tension record each time it is accessed. In the future we will experiment with more efficient ways of maintaining an individual transaction's statistics reports.

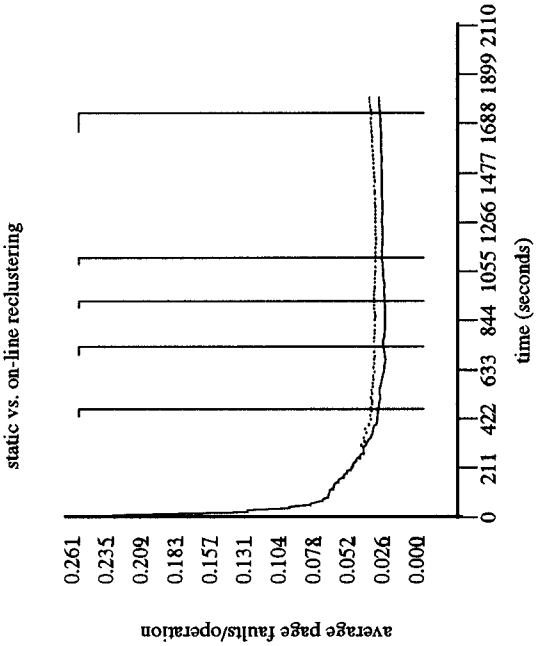

FIGURE 5. Opener, CAT = 0.630, RT = 0.883: normal workload.

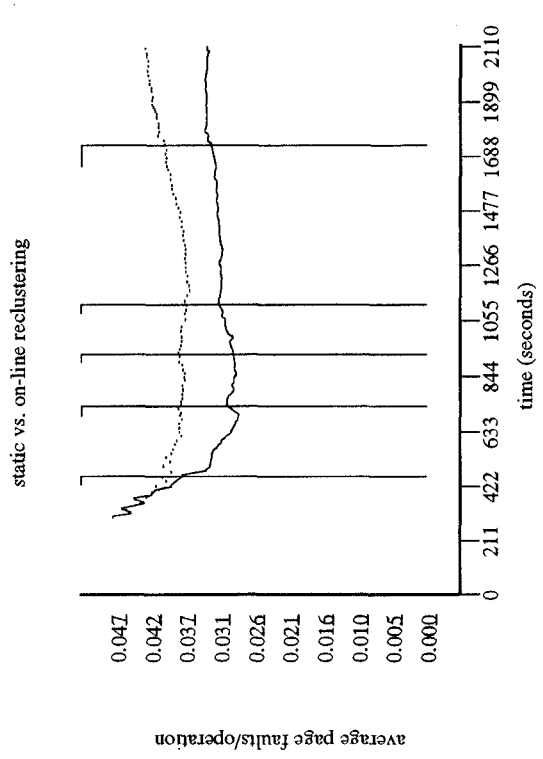

FIGURE 6. Closer, CAT = 0.630, RT = 0.883: normal workload.

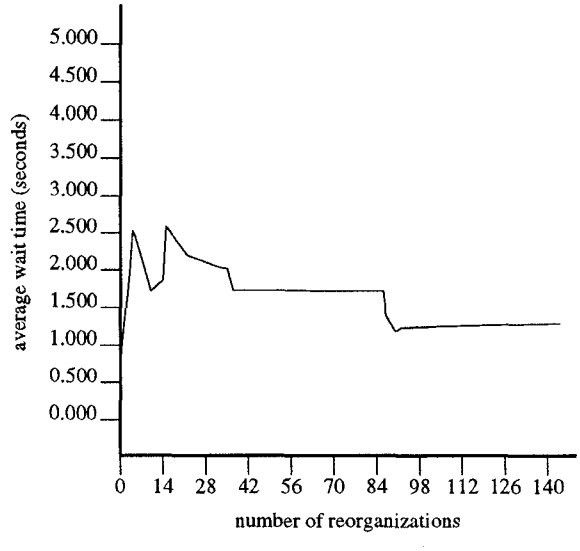

FIGURE 7. Wait Time vs. Reorganizations: Normal Workload

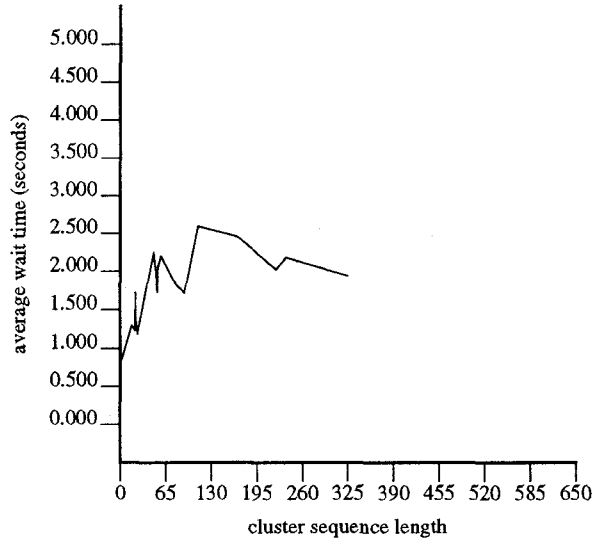

FIGURE 8. Wait Time vs. Cluster Sequence Length: Normal Workload

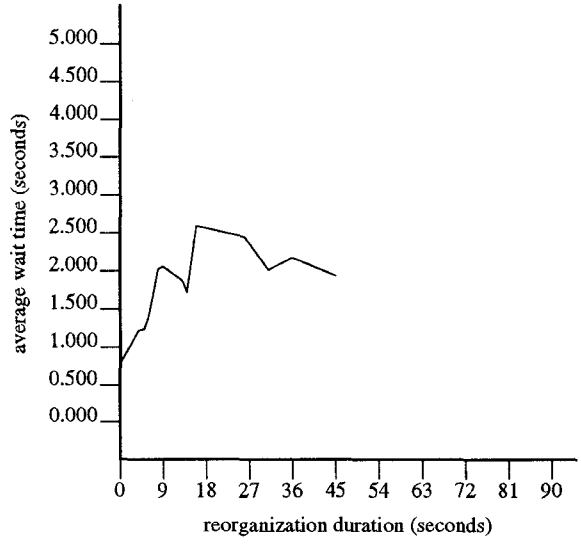

FIGURE 9. Wait Time vs. Reorganization Duration: Normal Workload

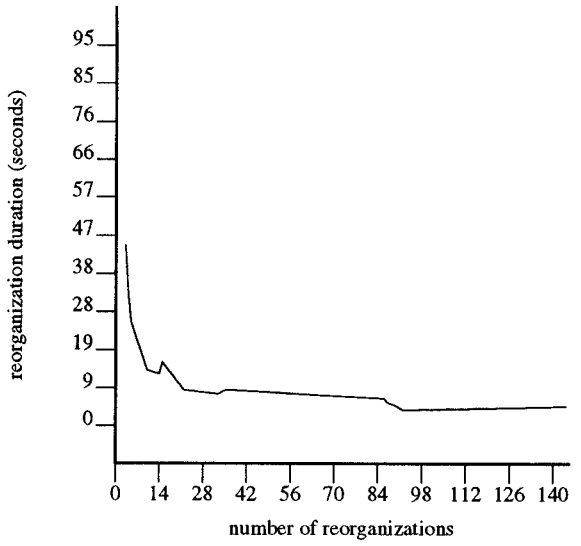

FIGURE 10. Reorganization Duration vs. Number of Reorganizations: Normal Workload

6 REDUCING TRANSACTION DELAY

In the previous approach to reorganization, the Reorganizer moved all of the objects in the current cluster sequence to their new locations in one transaction. It was shown that this delay was due in a major part to the Reorganizer's activities. In particular, the factors involved cluster sequence length and the duration of reorganizations.

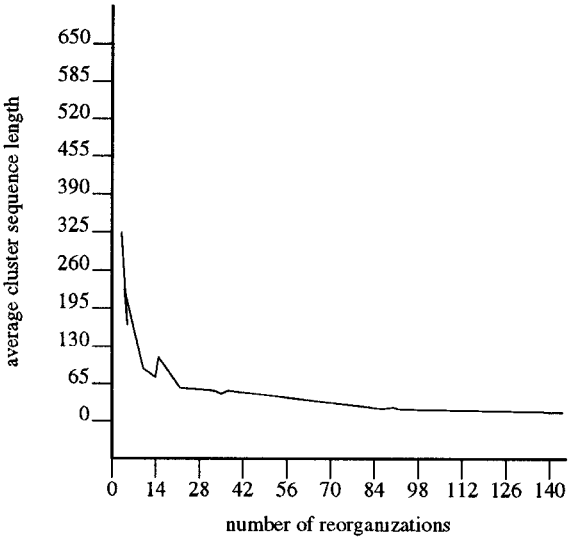

FIGURE 11. Cluster Sequence Length vs. Number of Reorganizations: Normal Workload

In an attempt to reduce the average transaction wait time, we performed an additional set of runs using the optimal CAT and RT threshold settings found in the experiment above, but with the exception that the Reorganizer was only allowed to process the current cluster sequence in increments of the average cluster sequence length experienced by the Reorganizer in the previous set of experiments. The length of the increment is called the *reorganization quantum*. For example, if the average cluster sequence length for a run was 50, the Reorganizer would process a cluster sequence of length 100 in two separate transactions, one being submitted immediately after the other.The rationale for this modification was to allow user transactions to be interleaved with a reorganization and thereby reduce the average wait times for user transactions. The main concern with this approach, obviously, was whether or not such a method would preserve any amount of reduction in the average miss rate observed for the corresponding run in the previous experiment.

For this experiment the average wait times for user transactions was reduced 26%, 23%, and 15% for the uniform, normal, and Zipfian workloads respectively. A surprising result was that for the uniform and normal workloads the average miss rate also decreased below that of the corresponding runs where the Reorganizer had an infinite reorganization quantum.

7 CONCLUSIONS

We have described a method for performing self-adaptive reclustering of complex objects in an on-line context. The problem of performing reclustering on-line was made more tractable by decomposing the fundamental tasks of a clustering method into concurrently operating components such that the determination of where objects should be placed (cluster analysis) and their actual placement (reorganization) could be de-coupled. Those components are: Statistics Collector, Cluster Analyzer, and Reorganizer. The cluster analysis algorithm used in by the Cluster Analyzer was designed to cluster complex objects that may be subjected to arbitrary combinations of set-oriented, materialization and navigational access methods. To guide the cluster analysis we employ database usage metrics called heat and tension which represent object references and co-references respectively. We presented rules for triggering cluster analyses and reorganizations to eliminate needless overhead by preventing cluster analyses and reorganizations from occurring where no significant changes to the database organization would be made.

We evaluated the effectiveness of our method using a benchmark that models an existing on-line transaction processing system. We showed that the cluster analysis and reorganization triggering rules that we defined could be effective in reducing the average miss rate for object accesses below that of runs where self-adaptive, on-line reclustering was not used. In general, a policy of setting conservative thresholds for both triggers was the most effective. The additional time and space overhead required to support on-line reclustering was shown to be modest. However, the introduction of on-line reclustering was shown to cause increases in the average transaction wait times due to the database lock contention caused by the Reorganizer. The increase in the average wait time was shown to be related to a combination of factors including the average cluster sequence length and the duration of reorganizations. To mitigate this problem we re-ran the optimal runs identified in the

main experiment with the modification that the Reorganizer processed the cluster sequence in increments, called a reorganization quantum, allowing user transactions to be interleaved with a series of Reorganizer transactions. This modification was shown to reduce the average wait time and in some cases was accompanied by an additional reduction in the average miss rate.

However, we know that more work must be done to evaluate this approach. In particular, we must examine the use of self-adaptive on-line reclustering in the presence of higher degrees of concurrent transactions. We must also evaluate its effectiveness on a wider range of applications types, and diversity of transaction mixes. Finally, we must evaluate it over a wider range of object sizes and degrees of structural interconnection.

ACKNOWLEDGEMENTS

We are grateful to Goetz Graefe for the many helpful ideas and intuitions he shared with us about this work.

REFERENCES

1. Bancilon, F., Delobel, C., Kanellakis, P. *Building An Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann, San Mateo, CA, 1992. p. 51.

2. Banerjee, J., Kim, W., Kim, S. J., Garza, J. F. "Clustering a DAG for CAD Databases", IEEE Transactions on Software Engineering, vol. 14, no. 11 (November 1988).

3. Cheng, J.R. and Hurson, A.R. "Effective Clustering of Complex Objects in Object-Oriented Databases," SIGMOD 1991, Denver, Colorado.

4. Drew, P., King, R., Hudson, S. "The Performance and Utility of the Cactis Implementation Algorithms", Proceedings of the VLDB Conference, 1990, Brisbane, Australia.

5. Gray, J., Putzolo, F. "The 5 Minute Rule for Trading Memory for Disc Accesses and The 10 Byte Rule for Trading Memory for CPU Time", Proceedings of the ACM SIGMOD Conference, San Francisco, CA. May 1987, pp. 395-398.

6. Harrus, G., Benzaken, V., Delobel C. "Measuring Performance of Clustering Strategies: the CluB-0 Benchmark", GIP-Altaïr-INRIA Technical Report 66-91, 1991.

7. Hudson, S. E., King, R. "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System", ACM Transactions on Database Systems, vol. 14, no. 3 (September 1989).

8. Horowitz, E., Sahni, S., *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, Maryland, 1978, pp 265-266.

9. Itasca System, Inc. *C API User Manual for Release 2.1*. 1992, p. 3-57.

10. Jackson, F., "Generation Scavenging: An Efficient, Unobtrusive, Portable Garbage Collector, Dr. Dobb's Journal, pp. 16-28, May 1990.

11. Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York, 1991. pp. 181, 190.

12. Knuth, D. *The Art of Computer Programming, Vol III: Sorting and Searching*. Addison-Wesley, Reading, MA., 1973.

13. Lieberman, H. and Hewitt, C. "A Real-Time Garbage Collector Based on the Lifetimes of Objects", *Communications of the ACM,* vol. 26, no. 6, June 1983, pp. 419-429.

14. McClenaghan, C. "OODBMS Benchmark Specification", U S West Advanced Technologies Technical Report, no. AT-12/99-001523-00.01, December 1991.

15. Object Design Inc. *ObjectStore User's Guide*. Burlington, Mass., 1992, pp. 63-65.

16. Objectivity Inc. *Objectivity/DB Release Notes*. Version 2.1, May 4, 1993, pp. 4-1,16,17.

17. Omiecinski, E. "Incremental File Reorganization Schemes", Proceedings of the VLDB Conference, 1985, Stockholm, Sweden, pp. 346-357.

18. Servio Logic Development Corporation. *Programming in OPAL*. Version 1.4, 1988, pp. 12-6,7; 17-1,2..

19. Schkolnick, M., "A Clustering Algorithm for Hierarchical Structures", ACM Transactions on Database Systems, vol. 2, no. 1 (March 1977), pp. 27-44.

20. Schwartz, M. *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison Wesley, Reading, MA, 1987, pp 24-30.

21. Thomasian, A. "Performance Limits of Two-Phase Locking", Proceedings of the 7th International Conference on Data Engineering, Kobe, Japan, pp. 426-435.

22. Tsangaris, M. M., Naughton, J. F. "A Stochastic Approach for Clustering in Object Bases". Proceedings of SIGMOD, Denver, Colorado, May 1991, pp. 12-21.

23. Weikum, G., "Clustering vs. Concurrency A Framework and a Case Study," MCC Technical Report, vol. ACA-ST-096-89, Austin, TX, February 1989.

24. Yu, C. T., Suen, C. M., Lam, K., Siu, M. K., "Adaptive Record Clustering", ACM Transactions on Database Systems, vol 10., no. 2 (June 1985), pp. 180-204.

25. Zipf, G. K., *Human Behavior and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, Reading, MA., 1949.