Clustering of Client-Sites in Three-Tier Database Architectures[*]

Je-Ho Park
Voicemate, Inc.
New York, NY 10014
jhpark@voicemate.com

Vinay Kanitkar
Akamai Technologies
Cambridge, MA 02139
kanitkar@akamai.com

R.N. Uma
Univ. of Texas at Dallas
Richardson, TX 75083
rnuma@utdallas.edu

Alex Delis
Univ. of Athens,
GR15784, Athens, Greece
ad@di.uoa.gr

September 2001

Abstract

Conventional two-tier databases have shown performance limitations in the presence of many concurrent clients. We propose logical grouping of clients (or clustering) as the means to improve the performance of collaborative networked databases. In this paper, we discuss a three-tier client-server database architecture (3t-CSD) featuring the above partitioning. The proposed clustering is based on the similarity of clients' access patterns. Each cluster is supervised by a designated manager that coordinates data sharing among its members. A number of clients is optimally partitioned if sites in each individual cluster have the maximum common data access probability possible. We initially show that the optimal client clustering problem is NP-complete and then we develop two approximate solutions based on abstraction and filtering of statistics for client access patterns. Our main goal is to compare the performance of the conventional and three-tier client-server database architecture with respect to the transaction turnaround times and object response times. After developing system prototypes that implement both two-tier and 3t-CSDs, we experimentally show that as long as good client-clustering is possible, the 3t-CSD architecture yields sizable gains over its conventional counterpart. We also compare and evaluate the effectiveness of the two proposed techniques used to create client clusters. Finally, we examine the role of several preprocessing schemes used to reduce the volume of the input data supplied to the clustering techniques.

Keywords: Collaborative Database Architectures, Multi-Tier Databases, Logical Clustering, Genetic Algorithms.

[*]This work was supported in part by the National Science Foundation under Grant NSF IIS-9733642 and CATT, a New York Office of Science, Technology and Academic Research (NYSTAR) designated Center for Advanced Technology in Telecommunications.

1 Introduction

Efficient management of high volumes of data in contemporary networked environments poses major challenges and offers new opportunities for improved performance rates[12, 13, 23, 24, 18]. In this regard, two-tier client-server databases (CSD) have pushed graphical user interfaces as well as partial database processing to client sites [26]. This trend has increased considerably as powerful networked PCs and workstations have become commodities [3, 8, 4]. Conventional two-tier CSDs have shown reduced response times for client transactions over their centralized counterparts. However, as servers are shared among many users, they become points of contention. This is known as the scalability problem and has been identified as a key research issue [1]. A number of existing proposals advocate the use of short-term memory caching at clients in order to yield better response times for database clients[3]. In [8], available client resources are used to stage server originating data in an inter-transaction fashion. High-speed networks have also created opportunities for the caches of other clients to be exploited as potential data sources [15]. Data request forwarding techniques that allow client requests to be satisfied by other clients (instead of servers) have been introduced in distributed file systems and CSDs [2, 7].

In an effort to approach the CSD scalability problem from a different direction, we examine the grouping of clients, that access similar sets of server-originating objects, into quasi-independent and cooperating clusters. The main goal of such clustering is to be able to satisfy as many data requests within the cluster as possible, thus, reducing the burden on the server(s) and improving overall system scalability. Therefore, it is essential that these distinct collaborative clusters be well-formed. Specifically, good client clustering solutions should satisfy the following properties:

1. The clients in each cluster should have as much of an overlap in their data accesses as possible. This is essential as a larger overlap implies a greater degree of sharing among the clients in a group. Hence, fewer objects have to be requested from outside the cluster.
2. Clients in any two separate groups should have a very small, ideally zero, overlap in their data requirements. This is because the coordination of all inter-cluster data accesses has to be performed by the server. A large number of such accesses can cause a considerable overhead on the server and thus reduce its scalability, in terms of the total number of clients that it can serve.

A designated site within each cluster –possibly a client machine– undertakes the role of a caching-agent between the clients and the server and is termed the *Intermediate Cluster Manager* (ICM). ICMs are connected to database server(s) and the interaction between clients and server(s) is performed with their assistance. The resulting three-tier alternative CSD architecture (3t–CSD) avoids performance degradation by delegating many time-consuming server tasks to this intermediate layer in the data access hierarchy.

We consider a cluster of clients to have been optimally formed if the participating sites in a cluster demonstrate the maximum possible similarity in terms of object accesses. In this paper, we show that optimal client clustering based on data access patterns is NP-complete. We, then, present two techniques to generate client clusters: a heuristic-based greedy method and another one based on the genetic algorithm (GA) approach [10]. The greedy method has the advantage of having a low computational cost as it consists of a single-pass [6]. The GA-based technique commences with a large population of possible solutions and performs a “parallel” (multi-modal) search in the solution space. The input to these clustering techniques is a compilation of the observed data/objects accesses by all clients. The volume of this information is directly proportional to database objects and participating clients as well. Subsequently, larger numbers of objects and clients imply very voluminous input data for the GA. Therefore, we employ two preprocessing schemes that can effectively reduce the data volume in discussion. In this regard, one of our key goals is to investigate the impact of the suggested preprocessing schemes on the quality of the produced clustering solutions. Client clustering is carried out in an off-line fashion [7, 27]. Client access patterns are transformed to bitmaps in order to enable the efficient functioning of the proposed algorithms. We also show that the way initial chromosome populations are generated is crucial to the quality of the formed client clusters. We use two such techniques for initialization: a heuristic based greedy method and a purely random one. From our experiments, it is shown that the latter consistently outperforms the former in creating good clustering results. Independent of the initialization method used, our experimental results indicate that the 3t–CSD does offer enhanced performance rates over the conventional two-tier CSD configuration. Two different workloads corresponding to realistic access patterns are used in deriving the above.

This paper is organized as follows. Section 2 discusses the proposed 3t-CSD cluster-based architecture and compares it with the conventional two-tier one. In Section 3, we describe the optimal client clustering problem and we show that it is NP-complete. Techniques to reduce the size of the used data to create client clusters and the heuristic algorithms used are discussed in Section 4 and 5 respectively. Section 6 presents our experimental goals and setting while Section 7 discusses our main experimental results. Related work and conclusions can be found in Section 8 and 9 respectively.

2 A Cluster-Based Database Architecture

This section outlines the *Logically-Clustered (3t-CSD)* database configuration whose novel concept is the *Intermediate Cluster Manager* (ICM). In this, the primary copy of the database is hosted by the server and clients communicate with it via IPC abstractions. User transactions are initiated at the client-sites. The required data/objects are fetched from the server and processed by client transaction managers. The latter use their main memory and disk space as cache buffers in order to maintain local copies of objects. The server performs only low-level database functionalities on behalf of requesting clients. In addition, the server provides serializable access to shared data by maintaining a global lock table and associated structures. Two types of object locks are permitted: *shared* (read-only) and *exclusive* (read-write) [20]. It is possible for several clients/transactions to obtain shared locks (SL) simultaneously on the same data item. However, exclusive locks (EL) are granted to only one client at a time. We assume that inter-transaction caching at the clients is permitted, i.e., cached database objects/locks can remain in client caches across transaction boundaries [19, 8].

Clients that demonstrate similar object access patterns are clustered into groups. Each such group is managed by an ICM which cooperates not only with the database server but also the clients. In essence, ICMs provide the necessary directory services that allow individual clients within clusters to share data. The resulting 3t-CSD collaborative architecture is depicted in Figure 1(a). This is in contrast to basic CS configuration (Figure 1(b)) where there is direct interaction between the clients and the server.

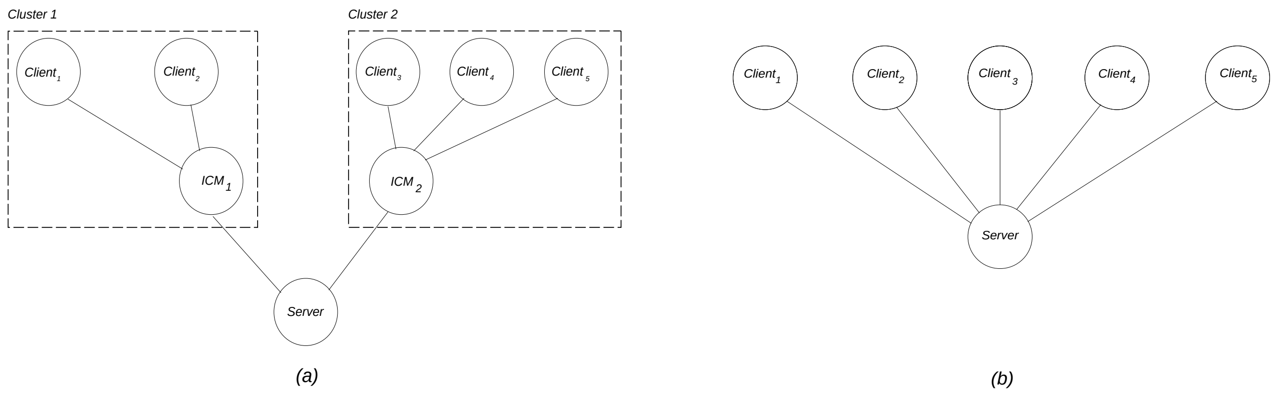


Figure 1: (a) Logically Clustered Database Architecture, (b) Client-Server Architecture

The 3t-CSD's server does not maintain a lock table for all clients in the system. Instead, it only maintains a lock table for locks granted to individual clusters, and each ICM maintains catalog information about objects and locks granted to the clients that constitute its cluster. When a transaction requests data objects/locks, the client first checks whether the requests can be satisfied from its local disk and memory caches. If an object is not available locally, then the client ships a request to its corresponding ICM. Depending on the type of requested lock (shared or exclusive), the ICM takes different steps to obtain it. The steps taken by the ICM for shared lock requests are shown below.

```
1. When an ICM receives an object request from a client, it looks up its cluster directory to see if another client
   within the same cluster has the object available in its cache.
2. IF (the object is present at a client within the cluster) THEN the ICM requests that client to forward a copy
   of the object to the requesting client as soon as possible.
3. ELSE ICM contacts the server and requests to ship a copy of it. The server can grant this request immedi-
   ately as long as clusters have not locked the requested object exclusively.
```

The steps that are taken when a client requests an EL are analogous to the ones described above. The basic requirement is that before a lock can be granted, it is necessary for all conflicting locks to be released.

In general, a client returns an object (and releases the lock on it) only when it receives a callback request for that object or when it needs to create free space in its cache. In either case, the client and the ICM have to ensure that the lock tables in the ICM and the server are updated correctly. The manner in which this is done depends on the type of lock that the client is about to release:

```
1. IF (the object has been updated at the client) THEN
    1.1 The latest version of the object is shipped to the server and the object is purged from the client's
        cache.
    1.2 Once the server has received the updated object, the client deletes the object and informs the $ICM$
        that it no longer maintains the object in its cache.
    1.3 The $ICM$ updates its cluster lock table to indicate that no copies of the object are present in its cluster.
2. ELSE_IF (the client has a SL on the object) THEN
    2.1 The client informs the $ICM$ that it is about to purge the object from its cache.
    2.2 IF(no other client in the cluster has cached the object) THEN the ICM informs the server that the
        object is no longer present in its cluster.
```

From the above description of the 3t-CSD, we can see that the use of ICMs offers tangible advantages over the two-tier client-server architecture in the following two situations: (i) when the data are cached at some client(s) in the cluster, the object location directory maintained by the ICMs allow requests for data to be satisfied without the intervention of the server, and (ii) the ICM's cluster-wide lock table allows sharing of data in its cluster and also guarantees that data accesses are serialized. Since interaction with the server is unnecessary in these situations, these benefits can contribute significantly towards reducing the load on the database server when the clients in each cluster have a considerable overlap in their data requirements. In such situations, the scalability of the server can be greatly improved compared to that of the server in two-tier systems [21]. The resource requirements of the ICMs are not very demanding either. Relatively small and inexpensive machines can be utilized to provide the object directory services and guarantee serialized data accesses.

The proposed 3t-CSD architecture based on logical client clustering severs the clients' direct connection to the server, and distributes the object-service and concurrency control effort over several intermediate servers. The performance of this 3t-CSD configuration, however, depends on the resulting partitioning of clients into clusters. We demonstrate this point with the help of an example.

Figure 2 depicts the case when two clients, C_1 and C_2, try to access the same object O_{91}. Assume that C_1 is updating O_{91}, i.e., it has an exclusive on it. If, at this point, a transaction at C_2 wants to access object O_{91}, then C_2 sends a request to the server for a copy of the object and the requisite lock. The server sends a callback request to C_1 asking it to return the updated object and release the lock. When C_1 is finished with its update, it releases its lock on O_{91} and ships the updated version to the server. This is now sent to C_2. Note that the server has to co-ordinate the movement of every object between any two sites in the system. This is precisely the factor which limits the scalability of the two-tier client-server configuration under conditions of low access locality and high contention.

Figure 3 shows a configuration of the cluster-based client-server system when clients C_1 and C_2 are in the same cluster. Consider the same situation as in the previous paragraph. Now, C_2 sends its request for O_{91} to the ICM in its own cluster. Since the ICM knows that C_1 is currently updating the object, it forwards the request for O_{91} to C_1. After the completion of its update, C_1 ships the object to the ICM which forwards it to C_2. Hence, the object request from C_2 is satisfied within the cluster without the server's involvement. The off-loading of such object requests from the server can greatly reduce the load on the database-server.

To make a case for optimizing the clustering of clients, we also consider a situation where clients C_1 and C_2 are

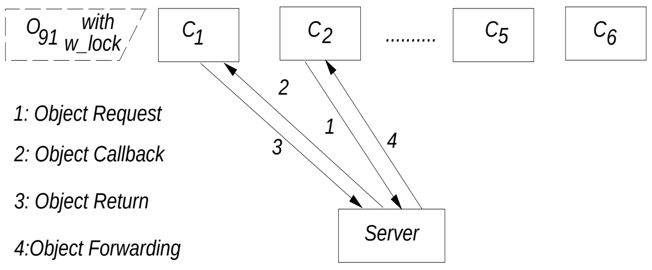


Figure 2: Object Request Satisfaction in a conventional CSD

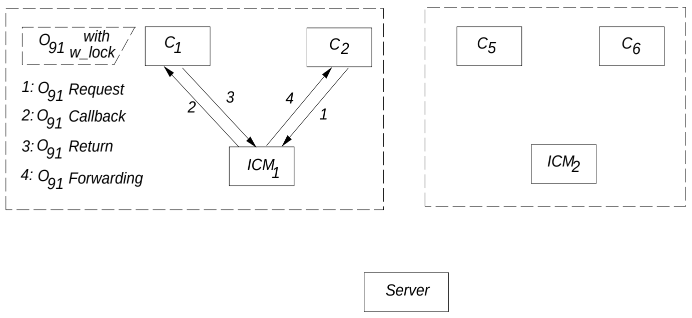


Figure 3: Object Request Satisfaction in Optimal 3t-CSD Case

in two different clusters. When C_2 sends the request for O_{91} to its ICM, the ICM has to forward the request to the server as no copy of the object is present within the cluster. After C_1 has completed its update, the server retrieves the object from C_1 and serves C_2's request by sending it the required object. The entire set of messages exchanged is shown in Figure 4. Therefore, when the clients are clustered without considering their data requirements, the 3t-CSD actually demonstrates worse performance than the two-tier CSD configuration.

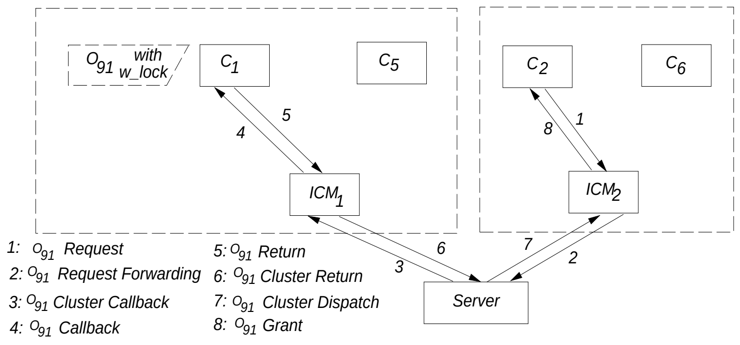


Figure 4: Object Request Satisfaction in Non-optimal 3t-CSD Case

Given the potential benefits of the three-tiered collaborative architecture, it is necessary that the clustering of clients is performed in the best possible way. Client clustering is considered to be optimal if participating clients in each cluster have the maximum possible common data accesses. In the next section, we define the optimal client-clustering problem in a formal way and show that it is NP-complete.

3 Optimal Client Clustering

The main goal of client-clustering is to colocate a number of clients in a cluster so that a good portion of object requests from a client can be satisfied by the other clients in the cluster. A good clustering of clients will avoid all unnecessary communication with the server. If the percentage of object requests that can be satisfied within the client clusters is significant then the system performance and scalability can be improved considerably. To increase the probability that an object request will be satisfied within a cluster, the clients in a cluster should cache the set of objects that are accessed by all of them. In the worst case, if a cluster is composed of clients which access (and cache) completely disjoint sets of objects, the probability of object request satisfaction within the cluster is zero. The Optimal Client Clustering problem is that of maximizing this probability.

3.1 Problem Formulation

Let $D = \{oid_1, ..., oid_z\}$ be the set of data objects in the shared database(s). Let O_i be the subset of objects in D that are accessed by a client i. When k clients are colocated in a cluster, the set of objects accessed by all the clients in that cluster is $CO = O_1 \cap ... \cap O_k$. The optimal client clustering problem is to find a subset of clients that allows the maximal $|CO|$ for a cluster of a given size.

Clients' data access patterns can be collected by monitoring their data requests over a period of time. To formalize the data access pattern, we use the following representation: Let C be a set of n binary bit-strings, one for each client, where the bit-string C_i, represents the data access pattern for client i. Let $C_i =< b_i^1, ..., b_i^z >$, where $b_i^j = 1$ if client i accesses the j^{th} database object, $b_i^j = 0$ otherwise. Using this representation of access patterns, the problem of finding a subset of clients with maximal common accesses becomes that of finding the largest set of overlapping 1's in the two-dimensional array C. To prove that the optimal client clustering is NP-complete, we first state it as a decision problem.

OPTIMAL CLIENT CLUSTERING Given C and two positive integers S and L such that $2 \leq S \leq n$ and $L \leq z$, does the set C contain a subset of S or more bit-strings such that the number of overlapping 1's among the strings in that subset is greater than or equal to L?

For the proof of NP-completeness of the client clustering problem, we show a reduction from an instance of the CLIQUE problem. In an undirected graph, a clique is a subset of its vertices that form a fully connected subgraph. Given a graph G, the problem of identifying a clique of a given size has been shown to be NP-complete [14, 16].

3.2 NP-Completeness of Optimal Client Clustering

Theorem: OPTIMAL CLIENT CLUSTERING is NP-complete.

Proof: We first show that the OPTIMAL CLIENT CLUSTERING problem belongs to the class NP. Then, we show that CLIQUE can be reduced to OPTIMAL CLIENT CLUSTERING in polynomial time. This will prove the theorem.

To demonstrate that OPTIMAL CLIENT CLUSTERING belongs to the class NP, we show that a certificate consisting of a subset C' of C and two integer values, S and L can be verified in polynomial time. The verifying algorithm first ascertains that the number of bit-strings in the given subset is greater or equal to S. In the second step, we calculate the number of overlapping ones among the bit-strings in C' and compare this number with L. This verification can be done in polynomial time. Therefore, we can say that OPTIMAL CLIENT CLUSTERING belongs to the class NP.

To prove that OPTIMAL CLIENT CLUSTERING is NP-hard, we show that CLIQUE can be reduced to OPTIMAL CLIENT CLUSTERING in polynomial time. Consider a graph $G = (V, E)$. Let G contain a CLIQUE of J ($J \leq |V|$) vertices. The corresponding instance of OPTIMAL CLIENT CLUSTERING has two parameters m and l, where m is randomly selected in the range $[2, J]$ and l is assigned the value $J - m$. This instance of OPTIMAL CLIENT CLUSTERING has $m \cdot |V|$ bit-strings, and the size of each bit-string is $l \cdot |V|$, where $m \cdot |V| = n$ and $l \cdot |V| = z$. The number of bit-strings in the maximal overlap is given by $J \cdot m$ and the size of the overlap is given by $J \cdot l$.

The transformation from an arbitrary instance of CLIQUE to a corresponding instance of OPTIMAL CLIENT CLUSTERING is performed in two phases. The first phase transforms the graph G in the instance of CLIQUE to the corresponding graph $G' = (V', E')$ in the instance of CLIENT CLUSTERING PROBLEM. In the second phase, the set of bit-strings (C') containing the maximal overlap is generated from the graph G'.

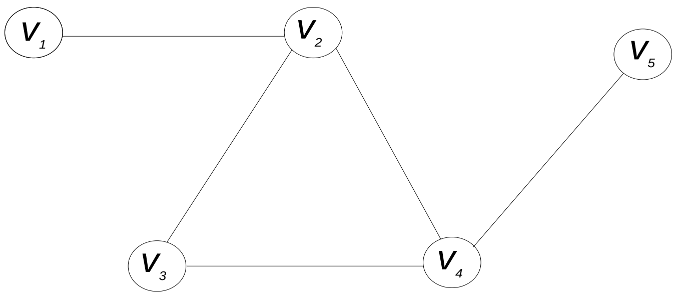


Figure 5: A graph with 5 vertices and a clique of size 3

Let $G = (V, E)$ contain a clique with J vertices (Figure 5 shows a graph containing a clique of size 3). The corresponding graph $G' = (V', E')$ is constructed as follows: in the first phase, to construct V', each vertex v_i in V is transformed into two sets of vertices: $RV_i = \{rv_i^1, ..., rv_i^m\}$ and $BV_i = \{bv_i^1, ..., bv_i^l\}$, ($m < J$ and $l = J - m$). To construct E', we connect each vertex in RV_i to all the vertices in BV_i. Therefore, there is now an edge between every vertex rv_i^r ($1 \leq r \leq m$) to every vertex bv_i^s ($1 \leq s \leq l$). The conversion of the vertex $v_1$is shown in Figure 6. Additionally, for every edge between v_i and v_j in E, for every pair of vertices (rv_i^r, bv_j^s) ($1 \leq r \leq m$ and $1 \leq s \leq l$), we add an edge in E'. The addition of edges to E' to replace the edge between v_1 and v_2 in E is illustrated in Figure 7. Converting all the edges and vertices of the graph G shown in Figure 5, we get the corresponding graph G' shown in Figure 8. This is the end of the first phase.

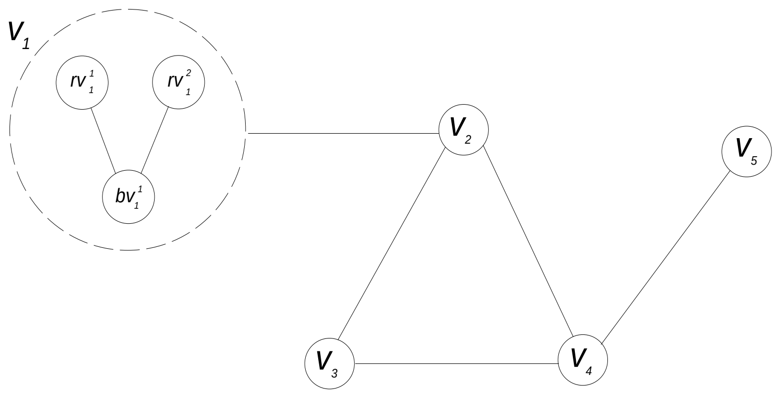


Figure 6: Vertex Conversion: V_1 is transformed into two groups of vertices, since $J = 3$ we can choose $m = 2 \; and \; l = 1$

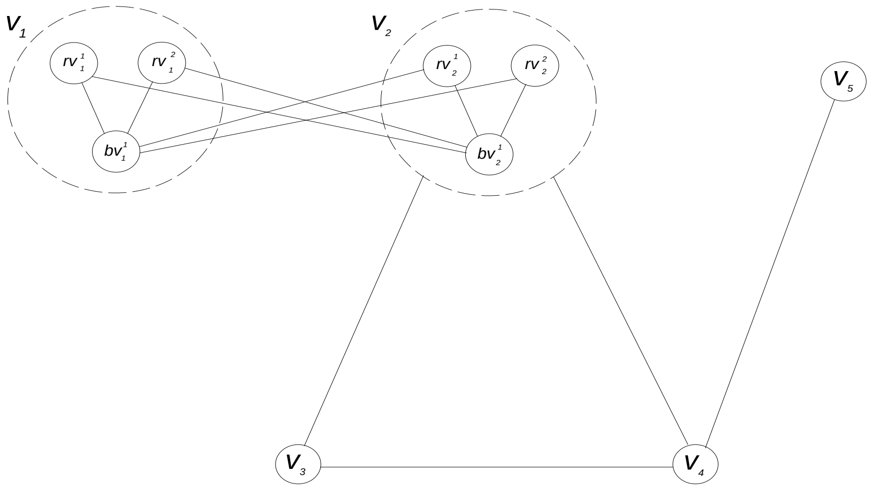


Figure 7: Addition of edges between RV_1 and BV_2, and RV_2 and BV_1 to replace the edge between V_1 and V_2 in the original graph

In the second phase, $m \cdot |V|$ bit-strings are generated from the graph G'. Each bit-string has $l \cdot |V|$ bits. For vertex rv_p^r, if there is an edge in E' between rv_p^r and bv_q^s ($1 \le p, q \le |V|$, $1 \le r \le m$ and $1 \le s \le l$), then the $((p-1) \cdot m + r)^{th}$ string has the $((q-1) \cdot l + s)^{th}$ bit set to 1. The matrix that is generated from the graph G' shown in Figure 8 is given below:

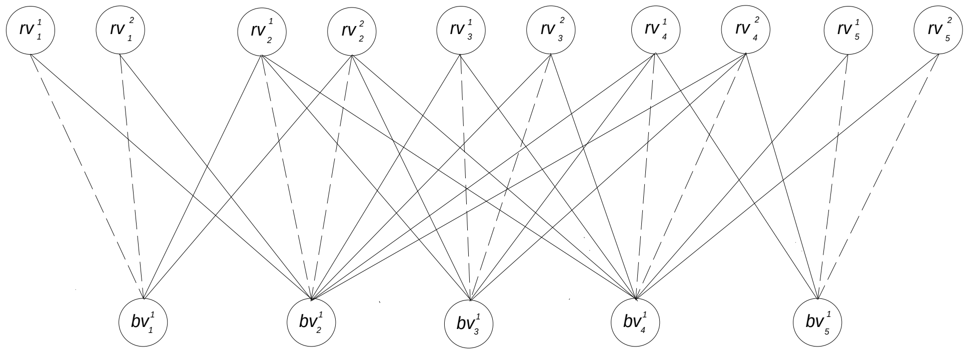


Figure 8: The graph G' after converting all the vertices in G and adding all required edges to E'

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Since the resultant graph G' from the graph $G = (V, E)$ has $|V'| = (m+l)\cdot|V|$ and $|E'| = m\cdot l\cdot(|V|+2\cdot|E|)$, it is clear that the transformation is done in polynomial time. If the original graph G includes a clique of size J that is composed of vertices $V1 = \{v_i, ..., v_{i+J-1}\}$, then the corresponding graph G' has two sets of vertices $RV = RV_i \cup RV_{i+1} \cup ... \cup RV_{i+J-2} \cup RV_{i+J-1}$ and $BV = BV_i \cup BV_{i+1} \cup ... \cup BV_{i+J-2} \cup BV_{i+J-1}$ such that all vertices in RV have the edges incident to all the vertices in BV. This implies that $J \cdot m$ bit-strings have exactly $J \cdot l$ overlapping 1's. This can be seen in the above matrix where 6 rows (from row 3 to 8) have 3 overlapping 1's.

Given a set C of n binary bit-strings, if S bit-strings show L overlapping 1's then the S bit-strings can be divided into $S/m = J$ groups of vertices regarding each group as one RV_i ($1 \leq i \leq J$). Similarly, the L overlapping 1's can be divided into $L/l = J$ groups of bits treating each group as a BV_i ($1 \leq i \leq J$). The existence of $m \cdot l$ edges between vertices in RV_i and BV_i induce a vertex v_i in the graph G. The presence of $m \cdot l$ edges between vertices in RV_i and BV_j, and $m \cdot l$ edges between vertices in RV_j and BV_i, implies that an edge exists between v_i and v_j in the graph G. Consequently, transforming each such set of bit-strings into J vertices induces a clique of size J in the graph G.

4 Pre-processing of Object Data Access Patterns

In this section, we describe the expected format of input data, the pre-processing techniques used to reduce its volume, and its representation when input to the clustering techniques. Similar pre-processing methods have been used in the study of web page access behavior [5]. For simplicity, we assume that a database is a collection of uniquely identifiable objects (via OIDs). The specific steps taken in order to prepare the data for the clustering techniques are shown in Figure 9. The first element in Figure 9 stands for the input data as collected by monitoring clients' behavior. Once this data is available, it is abstracted to a cell-based representation where each cell corresponds to a series of OIDs. This representation is then filtered so that only objects that demonstrate heavy traffic are retained for the clustering analysis. Before the latter commences, the representation is converted to a bitmap which allows for efficient identification of client clusters. The remainder of this section discusses each of the elements depicted in Figure 9.

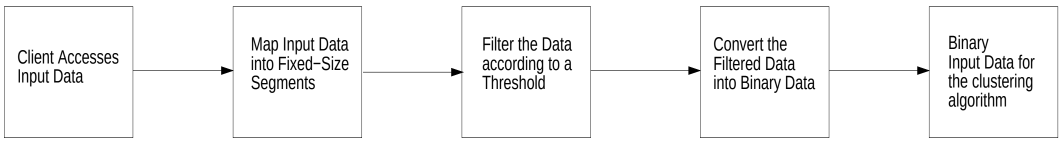

Client Accesses Input Data
Map Input Data into Fixed−Size Segments
Filter the Data according to a Threshold
Convert the Filtered Data into Binary Data
Binary Input Data for the clustering algorithm

Figure 9: Input Data Pre-processing

The raw input data is simply a count of the number of accesses made by each client to the objects in the database over a specific time interval. This data can be collected by monitoring the normal operation of the database system. We store this information in the form of a two-dimensional array where the rows correspond to the clients, and the columns represent the objects in the database. Therefore, for a system consisting of n clients and m objects in the database, the size of the array is $n \times m$. The array location $[i, j]$ stores the number of accesses made by Client i to Object j during the observation period. If we assume that a database consists of one hundred objects, a sample of the observed database access pattern for one client is shown in Figure 10. Here, the horizontal axis represents the individual database objects in increasing OID value and the vertical axis depicts the cumulative accesses for each object over an observation period.

As the number of objects in the database becomes larger, the size of the input rises drastically. This, in turn, causes the processing time and memory requirements of the client-clustering techniques to grow exponentially. Hence, it is necessary to devise techniques to reduce the size of the input without losing key characteristics in clients' access patterns. An elegant method of achieving this is to map each sequence of consecutively numbered objects into a single representative value. Such a summary representation can reduce the size of the input considerably. For example, in a database containing 10,000 objects, using a 10 to 1 mapping scheme reduces the number of individual access counts per client to 1,000. If one takes into consideration that related data objects are physically placed together, such a mapping technique is able to retain this property [17].

In our mapping schemes, we represent equal-sized sequences of object numbers by single values. Each such

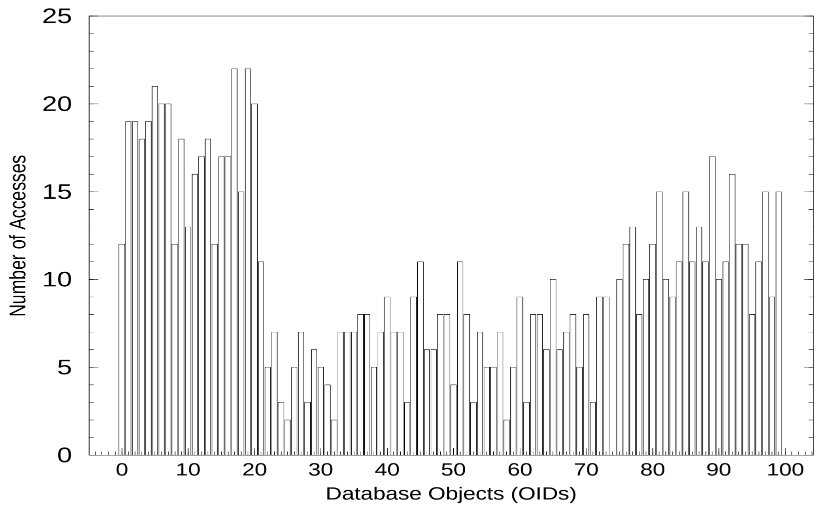


Figure 10: Observed Access Pattern for a Client

sequence of object numbers is called a *cell*. The access frequencies for the objects in a cell can be transformed into a single value using two measures:

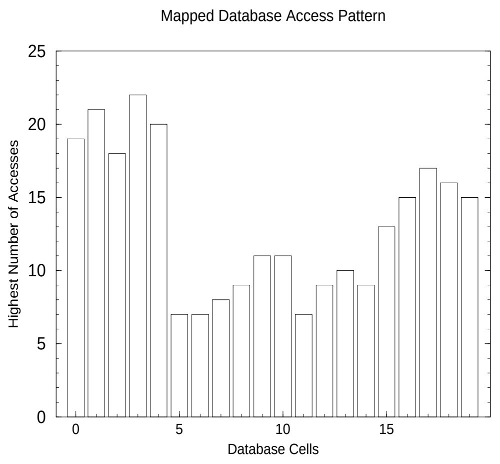
1. the highest access frequency; the object that has been accessed most frequently is treated as a representative for the entire cell, and
2. cell-cumulative; the sum of all object access frequencies in the cell.

For instance, consider the access pattern shown in Figure 10. Using a cell size of five, the mapping created by the highest frequency and the cell-cumulative frequency mapping techniques are shown in Figure 11 (a) and (b). Since each cell represents five objects in the database, the resulting number of values in the cell-based access pattern –along the x axis– is twenty. Our conjecture is that the cumulative representation contains a greater amount of information as it integrates all access frequencies in the cell.

Using the above access mapping techniques results in a loss of detail in each client's access information. Therefore, there is a trade-off between the granularity of the mapping used, and the reduction in the size of the input.

Once the frequency mapping has been computed, the access patterns are manipulated so that the more frequently accessed objects or database cells can be identified for each client. We apply a mean-based filtering to the compressed access frequency data. Here, the average of all cell access frequencies is calculated and values that are lower than the average are replaced by zeros. The effect of filtering the mappings shown in Figure 11 (a) and (b) are shown in Figure 12 (a) and (b) respectively.

As the two figures clearly indicate, the access patterns derived from the two mapping schemes followed by the filtering are not identical. In fact, unless the hot spots accessed by a client are very sharply defined, the two



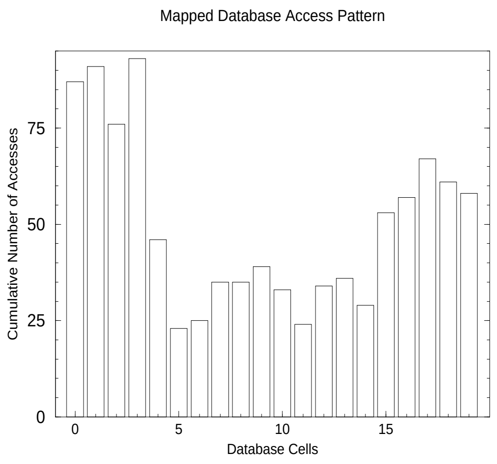


Figure 11: (a) 5:1 Highest Frequency Mapped Access Pattern, (b) 5:1 Cumulative Frequency Mapped Access Pattern

mappings, followed by filterings, will rarely identify the same hot sets.

Once the filtering phase is complete, the access patterns are transformed into a collection of binary bit strings. Here, the non-zero values in the filtered access patterns are represented by ones. We can say that formally each client's behavior is represented by such a bit string. Figure 13 shows the bitmapped patterns for the accesses depicted in Figure 12. For n clients, let C be the set of n binary bit-strings, where the bit-string C_i represents the data access pattern for client i. Hence, for client i, $C_i =< b_i^1, ..., b_i^z >$, where z is the number of distinct cells in the database(s). The bits in each C_i are set using the rule:

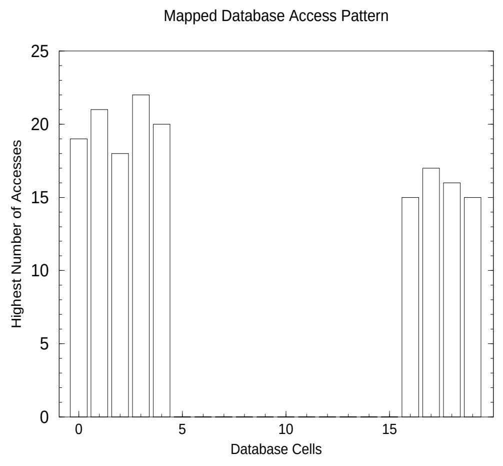
$$b_i^j = \begin{cases} 1, & \text{if the filtered access pattern for client } i \text{ has non-zero value for the } j^{th} \text{ cell} \\ 0, & \text{otherwise} \end{cases}$$

Using this representation of access patterns, the problem of finding a subset of clients with maximal common accesses becomes that of finding the largest set of overlapping 1's in the two-dimensional array C. In the next section, we describe two heuristic techniques that attempt to generate good client-clustering: a greedy technique and a genetic algorithm.

5 Techniques for Client Clustering

5.1 The Greedy Approach

The greedy approach uses the matching between the access patterns of clients as a clustering heuristic. Since this is a greedy algorithm, it makes the decision of allocating a client to a particular cluster based only on the best choice available at that instant. Clusters are assumed to have a maximum possible size, and once a cluster is *full* no more clients can be allocated to it. The outline of the algorithm is as follows:



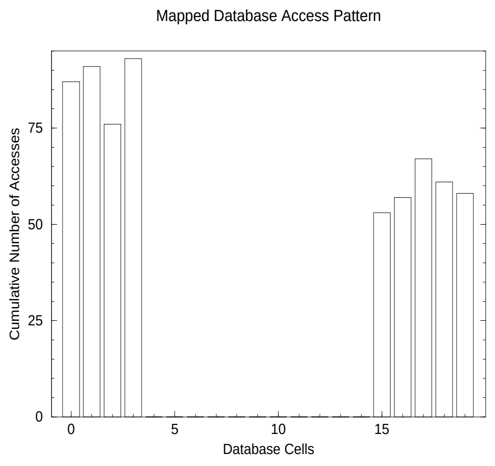


Figure 12: (a) Filtered Access Pattern (Mapped According to Highest Frequency), (b) Filtered Access Pattern (Mapped According to Cumulative Frequency)

1. For every pair of clients, the access patterns are compared and the number of overlapping accesses is maintained in a two-dimensional table.
2. Among the clients that are still unassigned to clusters, find (in the two-dimensional table) the pair of clients that have the greatest data accesses overlapped. This pair forms the basis of a new cluster.
3. For all unassigned clients: find the client with the largest non-zero overlap with this newly formed cluster (in Step 2), and add it to the cluster. Repeat Step 3 until there are no more clients that have an overlap with the new cluster or until the cluster is full.
4. After the members of the new cluster are determined, if the size of the cluster is small enough to be merged with one of the previously composed clusters without violating the maximum cluster size limitation then the two clusters are merged into one cluster. If there no clients with an overlap with any other client or cluster then go to Step 5, else go to Step 2.
5. Clients that do not have an access overlap with any existing clusters or unassigned clients are randomly distributed to those clusters that are still not full. If all existing clusters are full, new clusters are created for the still unassigned clients.

For n clients, m database cells, and p maximum number of clusters created during the execution of the program, the overall memory requirements can be summarized as: $(m \times n) + n^2 + n + (m \times p)$. The $(m \times n)$ factor is required to store the original data access patterns. Maintaining a table of clients' access pattern overlaps requires n^2 space, and n space is used to store current cluster assignments. Lastly, the $m \times p$ is used to store access overlaps of currently composed clusters. Therefore, this algorithm is quite efficient in terms of the required memory space. Furthermore, since this algorithm is a single-pass heuristic algorithm, the necessary CPU time is very small. However, it has several disadvantages that can make the generated solutions inadequate:

- It searches for a solution only in the local solution space, i.e., all future solutions are in the immediate neighborhood of the current point. This makes it susceptible to local minima, and
- It has no backtracking ability, i.e., once a client has been assigned to a cluster, it cannot be moved to another cluster later.

Considering the disadvantages of the greedy algorithm, we have considered using a dynamic programming ap-

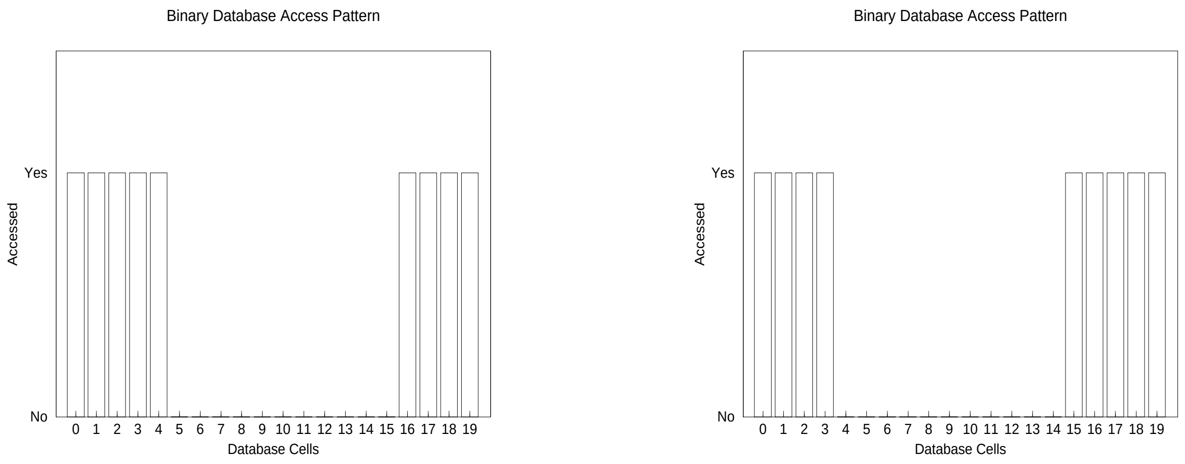


Figure 13: (a) Bitmapped Access Pattern (Mapped According to Highest Frequency), (b) Bitmapped Pattern Filtered (Mapped According to Cumulative Frequency)

proach. However, a dynamic programming technique is computationally very expensive [6]. Therefore, we opted for the use of a genetic algorithm (GA) that models the search of the solution space as an evolutionary process [9, 10, 28, 11].

5.2 Genetic Algorithm

A genetic algorithm has several advantages over other optimization techniques, namely:

- A genetic algorithm starts with a large population of feasible solutions and performs a parallel search of the solution space. This multi-modal approach makes the algorithm less likely to get caught in local minima, and ensures a more comprehensive search of the solution space.
- Only the pay-off (optimization function) information is used to guide the production of future generations. This makes the implementation of a GA easier than other search techniques which usually require complete information about the problem in order to generate and evaluate possible solutions.
- New solutions are generated using probabilistic transition rules instead of deterministic procedures. This allows a much more varied set of feasible solutions to be generated without resorting to randomized or exhaustive search techniques.

In the following subsections, we describe the encoding scheme used for the representation of genes and chromosomes. We then describe the fitness functions used to evaluate the chromosomes, explain how the initial population is composed, and present the used mutation and crossover operators. Finally, we provide an example using a small-sized clustering problem.

5.2.1 The Encoding Scheme for Candidate Clustering Solutions

In our GA, we represent each chromosome by a linked list of genes. Each gene in a chromosome represents a cluster of clients, and the chromosome itself represents the complete client clustering. This representation of each chromosome as a linked list of bit strings is different from general GA chromosome representations where the entire chromosome is one single bit string. We use this encoding since it makes the design of the required genetic operators much more straightforward. Now the destruction of clusters and the redistribution of clients among clusters can be implemented without changing the original semantics of the genetic operators.

5.2.2 Evaluation Parameters for a Client-Clustering Solution

Using an appropriate measure to judge the quality of a generated solution is very important if the GA is to converge upon the best achievable solutions. In designing a three-tier cluster architecture, there are two parameters, J and V, that we use to gauge the quality of the clustering formation.

J: Clients in a group should have a very high percentage of common data accesses so that most object requests can be satisfied within the cluster. The evaluation function J is a measure of these common data accesses of the clients in each cluster, taken over all clusters.

V: Inter-cluster data accesses need to be as few as possible. For every object request that necessitates lock callbacks and releases across clusters, the logical-clustering incurs a very high overhead. This second parameter, V, is the percentage of inter-cluster data accesses made by the clients in all the clusters.

Calculation of J: Consider a system with k clients. Let O_i be the set of objects accessed by Client C_i. For a possible clustering solution for this system, J is calculated as follows:

- Let the solution consists of 3 clusters, such that $Cluster_1$ contains clients 1 to 3, $Cluster_2$ contains clients 4 to b, and $Cluster_3$ consists of clients $b + 1$ to k.
- Let $Overlap_i$ be the set of objects (bits in the two-dimensional array) that overlap for $Cluster_i$. Therefore, for $Cluster_1$, $Overlap_1 = O_1 \cap O_2 \cap O_3$. The cardinality of $Overlap_i$ (i.e. $|Overlap_i|$) is the number of objects commonly accessed by clients in $Cluster_i$. The proportional overlap for $Cluster_i$ is $\frac{|Overlap_i|}{DBSize}$, where $DBSize$ is the total number of objects in the database.
- Given a clustering solution with r clusters, the overall evaluation function J is given by:

$$J = \sum_{i=1}^{r} \left(\frac{|Overlap_i|}{DBSize} \right)^2 + \frac{1}{r^2} \qquad (1)$$

In the evaluation function, the first part represents the potential data sharing among clients in clusters. The value for the first segment will change depending on the quality of clustering solution. Therefore, the first part of the function contributes to generation of good chromosomes that fulfill the objective concerning the quality of clusters. The evaluation function also encapsulates the number of clusters generated as a measure of quality of the clustering solution. This is an important parameter as it restricts the number of clusters generated. Without this restriction, the GA can generate solutions with an arbitrarily large number of clusters thus increasing the cost of implementing the ICM layer.

Calculation of V: Given a clustering solution, this function calculates a weighted average of the inter-cluster object accesses. Consider a system with k clients. Let O_i be the set of objects accessed by Client C_i.

- Let the proposed clustering solution consist of 3 clusters, such that $Cluster_1$ contains clients 1 to 3, $Cluster_2$ contains clients 4 to b, and $Cluster_3$ consists of clients $b+1$ to k.
- The set of distinct object accesses made by $Cluster_1$ is given by the union of the set of objects accessed by C_1, C_2 and C_3, i.e. $(U_1 = O_1 \cup O_2 \cup O_3)$. Let the cardinality of U_1 be N_1.
- The set of objects accessed by $Cluster_1$ that are also accessed by clusters 2 and 3 is given by: $(U_1 \cap [U_2 \cup U_3])$. Let the cardinality of the set generated by this formula be M_1.
- The proportion of objects accessed by $Cluster_1$ that are also accessed by $Cluster_2$ and/or $Cluster_3$ is M_1/N_1. Calculating these values for all clusters we have the proportion of objects also accessed by other clusters given by M_i/N_i for $Cluster_i$.
- The evaluation parameter, V, is calculated as the weighted average of the above values for each cluster. The formula, for r clusters, is:
$$V = \frac{\sum_{i=1}^{r} \frac{M_i}{N_i} \times c_i}{\sum_{i=1}^{r} c_i} \tag{2}$$
where c_i is the number of clients in $Cluster_i$. The c_i's are used to weight the evaluations of individual clusters so that a cluster with a larger number of clients will contribute more to the overall average.

We combine the two evaluation functions, J and V, into a single value. So as to assign comparable magnitudes to both of them, we introduce two integer multipliers α and β. The values for α and β are chosen in such a way that J and V are converted into the same order of magnitude. The combined evaluation is used in the GA to judge the fitness, f, of the chromosomes in each generation. The combined evaluation function is:

$$f = \frac{\alpha \cdot J + \beta \cdot (1 - V)}{2} \tag{3}$$

The $(1 - V)$ factor is necessary to convert V from a minimizing function to a maximizing function. The sum of two factors is divided by 2 in order to control the final value within the range [0,1]. Now, the final evaluation function evaluates the chromosomes concerning three factors: clustering quality, the number of generated clusters, and the inter-cluster data access pattern. From these factors, the chromosomes derived approach the optimal solution which maximizes the overlapping data access pattern while maintaining minimal number of clusters and interaction among client groups.

5.2.3 Generation of Initial Population

To generate an initial population, we use a simple heuristic algorithm. The initial population helps to accelerate the search for a feasible solution. Using this as our guiding principle, the algorithm we develop consists of the following steps:

```
1. IF(there are clients still unassigned to clusters) THEN
    1.1 Pick any one of these clients randomly.
    1.2 Find the cluster that this client has the greatest object access overlap with.
    1.3 IF(such a cluster is found) THEN add the client to that cluster.
    1.4 ELSE create a new cluster and add the client to it.
2. Repeat Step 1 until all clients have been assigned to clusters.
```

5.2.4 The Genetic Algorithm

The GA consists of the following steps:

1. Create an initial population of 100 chromosomes using the greedy algorithm described in Section 5.2.3. The genes in each chromosome represent clusters of clients. The greedy algorithm is based on maximal matching of bits in the clients' database access patterns against pre-existing clusters.
2. Calculate the fitness, f_i (Equation 3), of each member of the population and F which is the sum of all fitnesses. For each chromosome, its probability of selection for the next generation is calculated as $p_i = \frac{f_i}{F}$. In addition, the cumulative probability, q_i, is calculated which is given by $q_i = \sum_{j=1}^{i} p_j$.
3. Using the probabilities generated in Step 2, select the tentative members of the new population. Each member of the new population is picked by generating a random number $r \in [0, 1]$ and if $r < q_1$ then the first chromosome is selected, otherwise select the i^{th} chromosome such that $q_{i-1} < r < q_i$. One hundred chromosomes are picked by repeating this procedure. It is quite proper for some chromosomes (the more fit ones) to be picked more than once.
4. Using the chromosomes selected for the new population, we select the candidates for recombination arbitrarily. From this pool of candidate "parents", we select pairs of chromosomes randomly and generate their "children" by recombining the genes of the two parents. The two parents are then replaced by the child chromosome in the population. Finally, genes in randomly chosen chromosomes from the new generation are mutated.
5. Repeat Steps 2 to 4 for a pre-determined number of generations.

The memory required by the genetic algorithm is much larger than the greedy algorithm presented in Section 4.1. For n clients, m database cells, p as the maximum number of clusters in a generated solution, and q chromosomes in each generation, the memory requirements of the genetic algorithm can be written as: $(m \times n) + q \times (p \times (m + n))$. The first term in this expression, namely $(m \times n)$, is the same as the greedy algorithm, and the second term specifies the maximum amount of memory space that would be required to maintain one generation of solutions (chromosomes). What can make the second term very large is that for the genetic algorithm to provide a reasonable search of the solution space, the number of chromosomes, q, in each generation has to be large. In our experiments, the value of q is of the same order as the number of clients (n).

5.2.5 The Used Mutation and Crossover Operators

In the mutation, the clients from randomly selected cluster(s) are redistributed to other clusters using the algorithm described in Section 5.2.3. The mutation operator is used to create a set of alternative feasible solutions by destroying existing clusters and distributing the clients in them to other clusters. An example of the operation of the mutation operator is shown in Figure 14. In the first line, we show a chromosome representing a possible clustering solution. This chromosome consists of four genes each of which represents one cluster. Gene 2 is randomly chosen for destruction, and the clients in it are redistributed to the other clusters using the algorithm from Section 5.2.3. In this example, Client 4 gets assigned to Gene 0 but Client 9, which does not have an overlap with any existing genes (clusters), is moved to a separate gene (cluster).

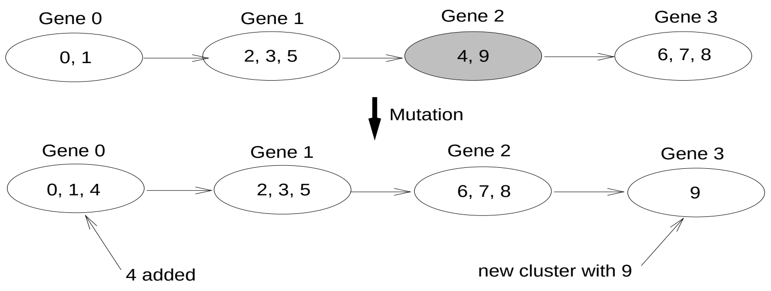


Figure 14: An Example of the Mutation Operation

For the crossover, randomly selected genes in two parent chromosomes are switched. Consider the example shown in Figure 15. The first part of the figure shows two chromosomes. The genes in the second chromosome are shown with thick borders. In the crossover, Gene 1 from the first chromosome is substituted with Genes 1 and 2 from the second chromosome. If the newly inserted genes cause a duplication of client(s) in the chromosome, then the original clusters containing the duplicates are destroyed and the non-duplicated clients from those clusters are redistributed (using the algorithm in Section 5.2.3). This is shown in part (c) of Figure 15. Client 1 is duplicated in the first chromosome. In this case, Gene 0 is destroyed and its constituent non-duplicated clients (Client 0) are redistributed to the other genes.

In some cases, clients can be potentially removed from a chromosome during the crossover opertion (e.g., Client 2 from Chromosome 1). Such clients need to be reintroduced into the existing clusters, including the newly created clusters (genes).

The logical clustering solutions generated by 12 generations of the above genetic algorithm are shown in Figure 16. The input to this example were the database access patterns of 10 clients. These access patterns were created randomly for a database containing 10,000 objects. The initial population of chromosomes was generated using the greedy maximal overlap algorithm. This initial population is shown as Generation 0 in Figure 16. Here, we can see that Chromosome 1 is rated as the best solution and Chromosome 3 is the worst.

After 12 generations of recombinations and mutations, we can see that the chromosomes are converging to-

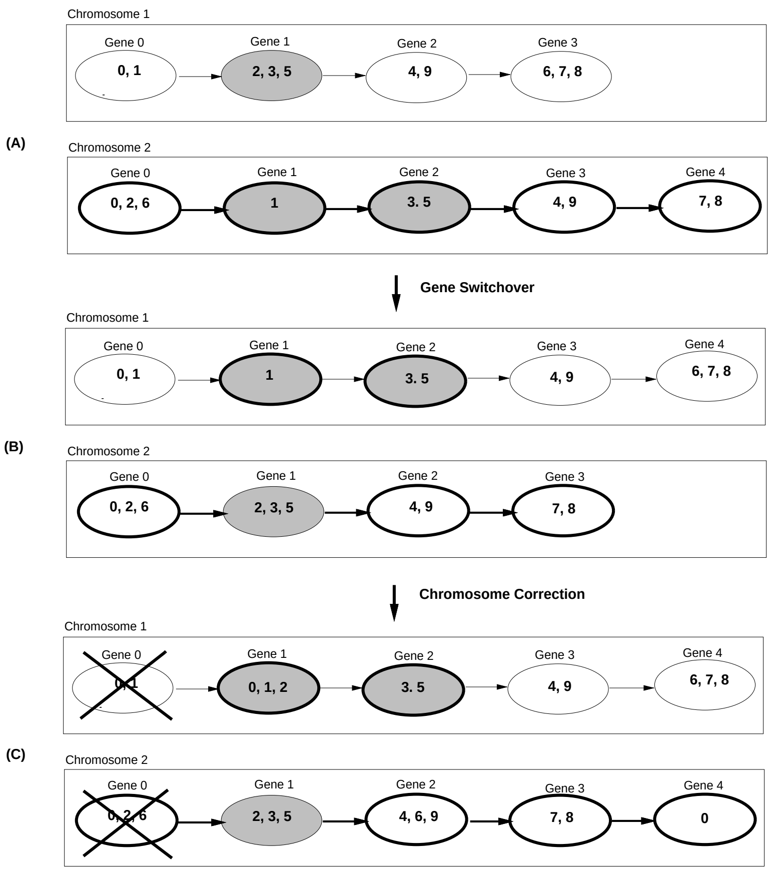


Figure 15: An Example of the Crossover Operation

wards a single optimal solution (according to the combined measure). Two of the four chromosomes are the same with a combined evaluation value of 0.5314. After a few more generations, all the chromosomes will suggest the same clustering solution. Since the GA is not an unimodal optimization algorithm, it can be seen that the final population also contains several unacceptable clustering solutions. In fact, this is an advantage of a GA. Since the search is multi-modal, a GA is less likely to get stuck in local optima.

6 Study Objectives and Experimental Testbeds

So far, we have outlined techniques to formulate client clusters in an off-line fashion by exploiting observed object access patterns. In this section, we discuss an experimental evaluation of the proposed logical client-clustering solutions and their effect on the 3t-CSD's performance indicators. We also carry out a sensitivity analysis of the different object mapping schemes used and investigate the role of genetic evaluation metrics in the derivation of viable client-clusters. More specifically, the main goals of our experimental evaluation are:

1. To investigate whether there is substantial gains in using a cluster-based 3t-CSD configuration over the

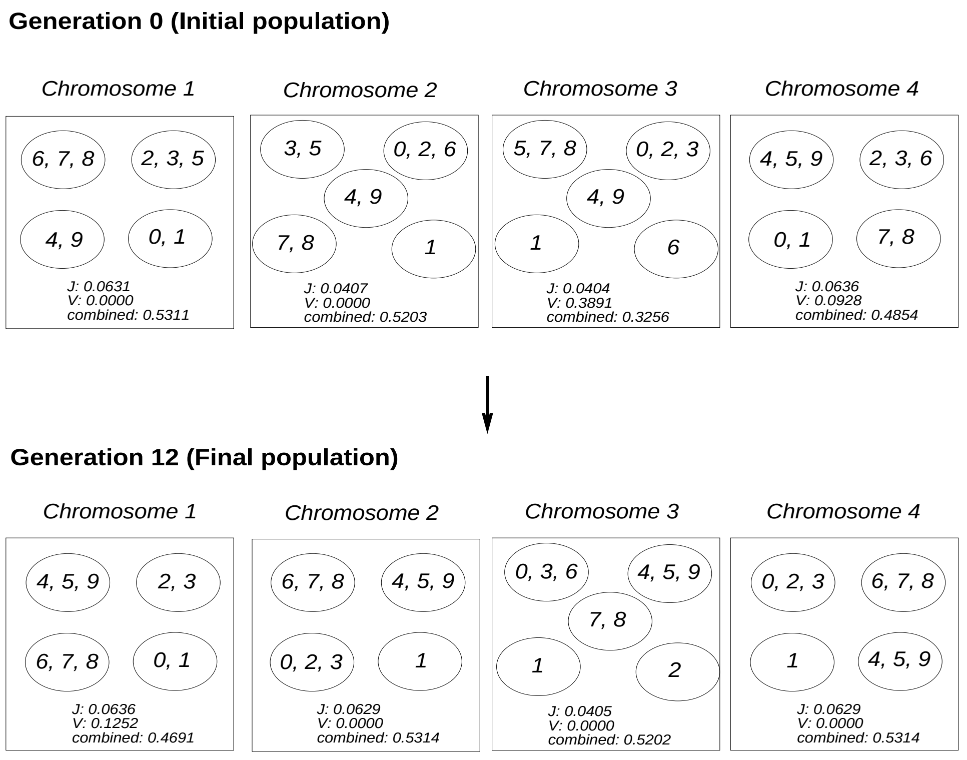


Figure 16: A Sample Run of the Genetic Algorithm with 10 Clients

traditional two-tier CSD. Our main hypothesis is that good clustering can effectively minimize interactions not only with the server but also with elements in other groups.

2. To examine the effect that the two techniques used to initialize the GA's first generation of solutions had on the quality of the generated clustering solutions.

3. To understand the effect of the different mapping schemes and granularities on the clustering results and, therefore, the efficiency of the 3t-CSD.

4. To study how three different genetic evaluation metrics affect the performance of the 3t-CSD for a varying number of clients.

We have developed full operational prototypes for the 3t-CSD as well as the baseline conventional CSD system. They both run in a network of six Sun ULTRA-1s workstations running Solaris 2.7 and connected via a 100-Mbps FastEthernet LAN. For our experiments, we assume a computing environment consisting of one database server and a varying number of clients.

We have created a page-mapped I/O software layer to manage object databases at both client, server and ICM sites. This layer allows for basic paged reads/writes and buffering. We assume that exactly one object is stored per page without loss of generality. In our experiments, we designate an individual workstation as the server site while we distribute a varying number of client databases equally over the remaining machines. Communication between the clients, formed ICMs and the server was done using TCP/IP sockets. The servers in both systems have been designed as connection-oriented programs, i.e., once a connection is established with an ICM or a client then that connection is maintained for the duration of the experiment. The ICMs are also connection-oriented, concurrent programs which maintain their socket connections with the clients and server throughout the execution of the experiments. This design decision was taken as the overhead incurred in the continuous creation and destruction of connections turned out to be cost-inefficient.

We have developed the ICM sites as multi-threaded programs where one thread corresponds to every client in the cluster. ICMs use their main memory to maintain the object directory and lock table for their respective clusters. Clients can also accommodate multiple concurrent requests using multi-threading on their own and use our memory-mapped I/O library to store in both its short and long term memory data objects. All packages have been implemented in C using the Solaris thread and socket libraries. Synchronization between the various threads within each process, when accessing global variables, is done using the Solaris mutual exclusion (mutex) functions. The values used for system parameters are listed in Table 1.

Parameter	*Experimental Value*
Database Size	10,000 objects
Server Main Memory Size	2,500 objects
ICM Main Memory Size	500 objects
ICM Disk Capacity	500 objects
Client Disk Cache Size	200 objects
Client Memory Cache Size	100 objects
Maximum Number of Objects Accessed by a Transaction	10

Table 1: Experimental Parameters

In the two-tier CSD, the server processes all requests for data objects and locks from the clients. It maintains an up-to-date lock table and resolves all concurrency issues. In order to do this efficiently, we have designed the server as a multi-threaded process that assigns one thread to each client in the system. This thread is responsible for handling all future interaction with that client. The multi-threaded implementation of the server allows it to satisfy client object requests concurrently.

Transaction arrivals at each client are generated as a Poisson process with an fixed inter-arrival mean of three seconds. The processing time for each transaction is generated using an Exponential distribution which also has a mean of 0.5 seconds. The transaction load on the system is varied by increasing or decreasing the number of clients. Each transaction can request up to ten database objects; when these requested objects becomes available the transaction can be executed. The exact number of objects requested by each transaction is generated randomly. Transactions are executed as separate thread processes. If the required data is available at the client then it is locked by the transaction. If the object is to be updated then it marked as dirty so that it is written back when the transaction commits. Now, the transaction spends its prescribed processing time calculating products of random numbers in a loop. Once this loop terminates, the transaction releases its lock on the data and commits.

In our database access patterns, the hot area of the database is one half of the total database segmented as fifty disjoint hot spots [25]. The size of each hot spot is equal to 1% of the database size. Each client can access up to five of these hot spots, and 90% of the client transactions' object requests are made to those hot spots. The other 10% of a client's object accesses are made to the non hot spot region (which is also 50% of the database). We create two different workloads by changing the way in which the hot spots accessed by each client are selected. In the first workload, each client selects its hot spots randomly (out of the available fifty). We call this workload $W3$. In the second workload, called $W4$, each client picks a random set of hot spots from a pre-defined range of such areas. The first 10% of the clients select their hot spots between the range zero to four, the next 10% clients chose theirs between the range five to nine, and so on. These database access patterns are based on the HOTCOLD scenario presented in [3].

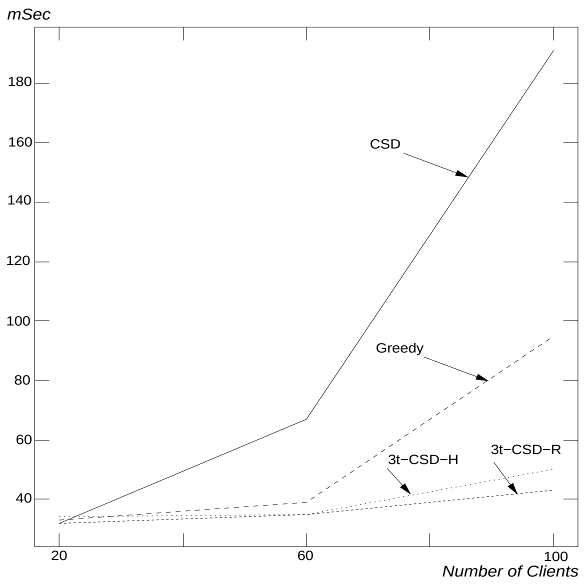
7 Experiments with Prototypes

7.1 Baseline Results

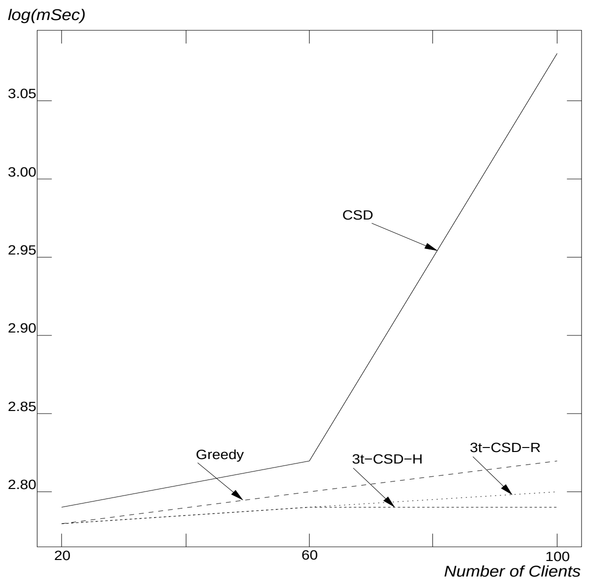
In our experiments, we measure the average object response times and the mean transaction turnaround times as the key performance indicators of the two-tier CSD and 3t-CSD configurations. As mentioned, we have experimented with two different methods of initializing the first generation of chromosomes of the genetic algorithm. In the following, we denote the clustering solution generated by the GA when using the randomized initialization as 3t-CSD-R. The clustering generated by the GA when the heuristic-based initialization is performed is termed as 3t-CSD-H. For the first set of experiments ($W3$ workload), the average object response times and transaction turnaround times for the basic CSD, 3t-CSD-R, and the 3t-CSD-H are shown in Figure 17(a) and (b) respectively.

Although, the performance of the two-tier CSD is comparable to the 3t-CSD configurations for 20 clients, it is apparent that the scalability of the two-tier CSD is severely limited. As the number of clients in the system is increased, the response times for clients' object requests increases very rapidly. The transaction turnaround times go up correspondingly. In the 3t-CSD architectures, the use of ICMs to off-load server tasks results in significantly lower object response and transaction turnaround times. This is because a considerable percentage of clients' requests can be satisfied within the cluster. Concurrency control and access serialization for the objects resident in a cluster are performed by the ICMs. The server is only required to co-ordinate inter-cluster object requests, i.e., when an object requested by a client has been locked in a conflicting mode by a client in another cluster. This is precisely the situation that the genetic algorithm tries to avoid – by clustering clients that access the same data together.

Unexpectedly, we observed that the 3t-CSD-R configuration demonstrates lower object response times than the 3t-CSD-H clustering configuration. This result is surprising because the heuristic-based initialization routine



(a) Average Object Response Times



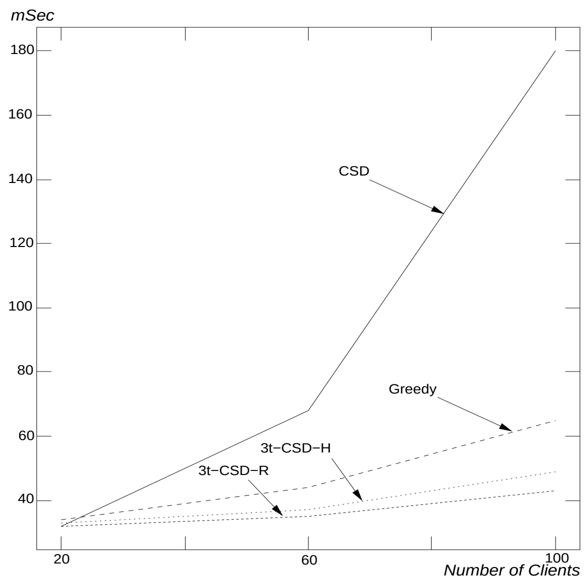
(b) Average Transaction Turnaround Times

Figure 17: Base Line: Experimental Set 1 – Workload $W3$

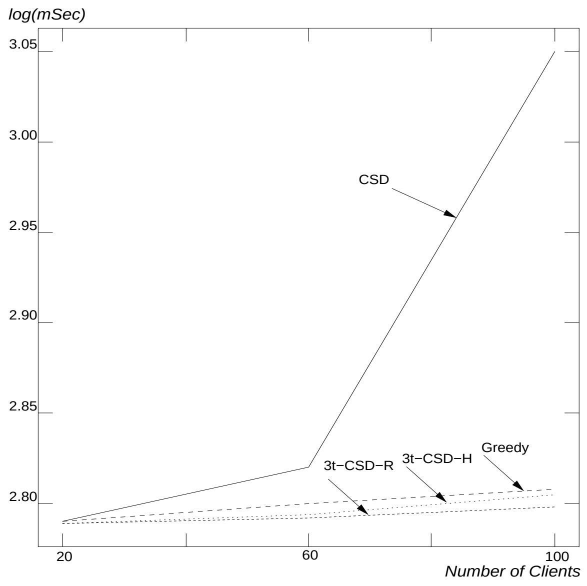
uses the access overlap as its metric for deciding which cluster to insert a client into. This is also the evaluation metric used by the genetic algorithm to decide which candidate clustering solutions are better than others. On the other hand, the randomized initialization procedure places clients into arbitrarily selected clusters.

It is our conjecture that the initialization performed by the randomized procedure will, in general, be better than that performed by the heuristic initialization routine. This is because the randomized routine is able to generate a highly varied set of clustering configurations. This initial generation is used by the genetic algorithm to create succeeding populations of chromosomes (clustering solutions), thus resulting in an extremely diverse coverage of the complete search space. In contrast, the greedy heuristic-based initialization results in a large number of very similar solutions. Due to this the GA has very little scope for generating diversity in its solution search space. Consequently, a very small number of unique solutions can be generated even after repeated applications of the mutation and recombination operators.

The results for Experiment Set 2 ($W4$ workload) are shown in Figure 18. The logically clustered configurations are able to demonstrate a much better level of performance than the two-tier CSD. The results follow the trends observed in the previous experiment to a great degree. One noticeable change is the smaller difference between the average object response times in the 3t-CSD configurations. This is due to the fact that the $W4$ workload is designed so as to make optimal client clusters easier to identify. Here, the hot spots for individual groups of clients are selected from within a particular set of hot spots. Consequently, the quality of the clustering configurations generated by the GA is almost identical irrespective of the initialization technique used.



(a) Average Object Response Times



(b) Average Transaction Turnaround Times

Figure 18: Base Line: Experimental Set 2 – Workload $W4$

7.2 Sensitivity Analysis

In the first set of these experiments, we evaluate the effect of varying mapping granularities on the results of the two clustering techniques. Our objective in doing this was to study the trade-off between the available detail in clients' access patterns and the corresponding performance of the clustering algorithms (and the 3t-CSD architecture). In the second set of experiments, we varied the evaluation metric used by the genetic algorithm to gauge the quality of client clusters. We did this in order to analyze the effect of our contrasting metrics on the clustering solutions. We also investigate the effect of using the $W3$ and $W4$ workloads to generate client data access patterns.

Experiment Set 1: We started by using the input at its original detail, i.e., at a 1:1 mapping. After this we increased the mapping granularity up to 40:1 in steps of 20. The number of clients in the system were 100, and the update selectivity of the transactions was 5%. We measure the performance of the 3t-CSD configuration using the average response time for object requests as our primary metric. This is because the delays encountered in obtaining requisite data have been observed to be the key factor affecting the transaction throughput. The genetic algorithm was run for a total of 50 generations. The time taken by the genetic and the single-pass greedy algorithm to form the client clusters are shown in Table 2. Unlike the GA, the time required for the greedy algorithm is directly proportional to the size of the input.

The average response times of clients' object requests for the $W3$ workload are shown in Figure 19(a). As the graph indicates, the object response times for the genetic algorithm are very much lower than those for the greedy algorithm when the observed object access frequency data is used as is. Consequently, the clustering generated

Mapping	*Time (seconds)*	
	Genetic Algorithm	*Greedy Algorithm*
1:1	7841	180
20:1	999	9
40:1	524	5

Table 2: Time taken by the clustering techniques

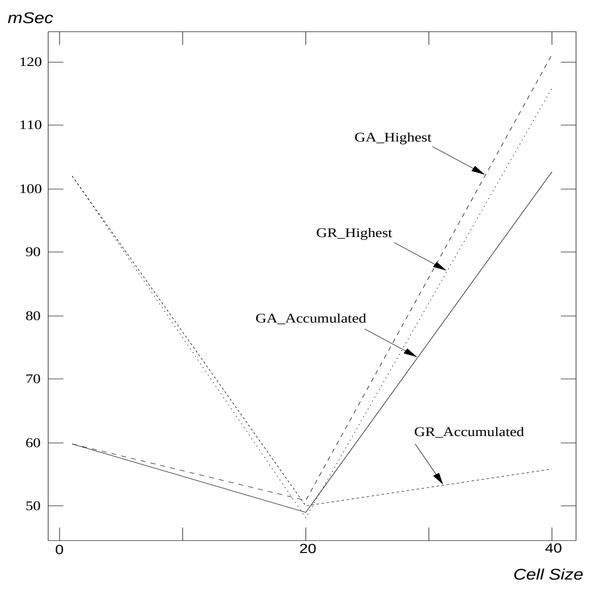
by the GA is able to demonstrate shorter transaction turnaround times (Figure 19(b)). This is because the genetic algorithm is able to start with the clusters generated by the greedy algorithm and further improve them. Since the mapping is 1:1, there is no difference between the height-based and cell-cumulative mapping schemes. However, once the mapping granularity is increased, the difference between the object response times for the two schemes becomes apparent.

At a 20:1 mapping, a vastly reduced level of detail about clients' access behavior is available to the clustering techniques. However, this actually proves to be beneficial, and now the clustering techniques are no longer confused by the very low level details about clients' data accesses. Hot spots in each clients' accesses can be identified and matched much more easily. This results in enhanced clustering solutions.

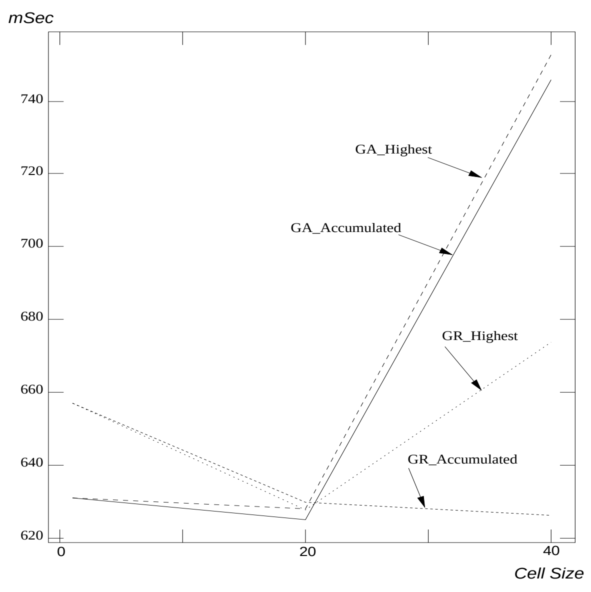
As the results show, changing the mapping to 40:1 leads to a deterioration in the object response times and transaction turnaround times for the clusterings generated by both clustering techniques. At this mapping granularity, the data access patterns become very fuzzy and it is very difficult to identify clients' hot spots distinctly. This is particularly true for the genetic algorithm whose evaluation metrics are highly sensitive to the available detail in clients' data accesses. Therefore, the degradation in the performance of the genetic algorithm is therefore considerably worse than that of the greedy algorithm.

For the $W4$ database access workload, the average object response times are shown in Figure 20(a). As the figure indicates, up to a mapping of 20:1 the results follow a similar trend as in the $W3$ access scenario. For a 1:1 mapping, the genetic algorithm is able to generate a better clustering of clients than the greedy algorithm. At a 20:1 mapping granularity, the performance of the two clusterings are very similar. However, at a 40:1 mapping, the 3t-CSD configuration suggested by the greedy algorithm actually performs better than the one suggested by the genetic algorithm. It is our conjecture that this is due to the hot spots for groups of clients being generated from within a specific range. The lack of detail in clients' data access patterns now causes contiguous hot areas to merge. This helps the greedy algorithm as it uses only the data access overlap to group clients. The average transaction turnaround times for the $W4$ scenario conform to the same trends as seen in the object response times (Figure 20(b)).

Experiment Set 2: In this set of experiments, we evaluate the performance of the 3t-CSD architectures generated by the genetic algorithm with three different evaluation metrics. As seen in Experiment Set 1, the trends observed in the transaction turnaround times closely follow those of the average object response times. Hence, we depict

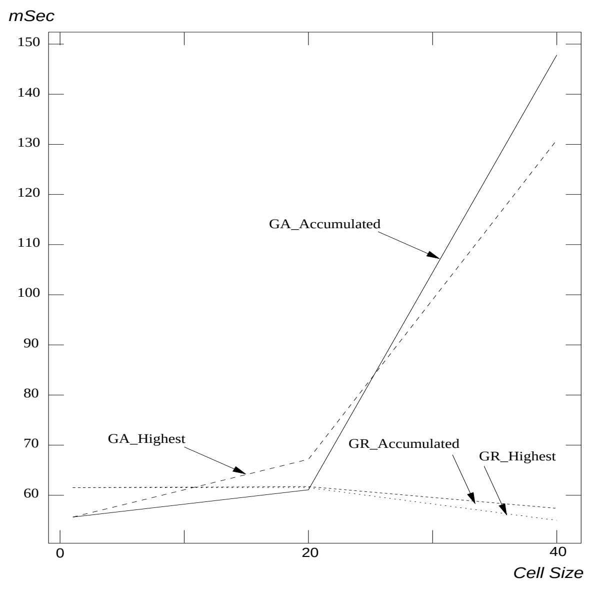


(a) Average Object Response Times

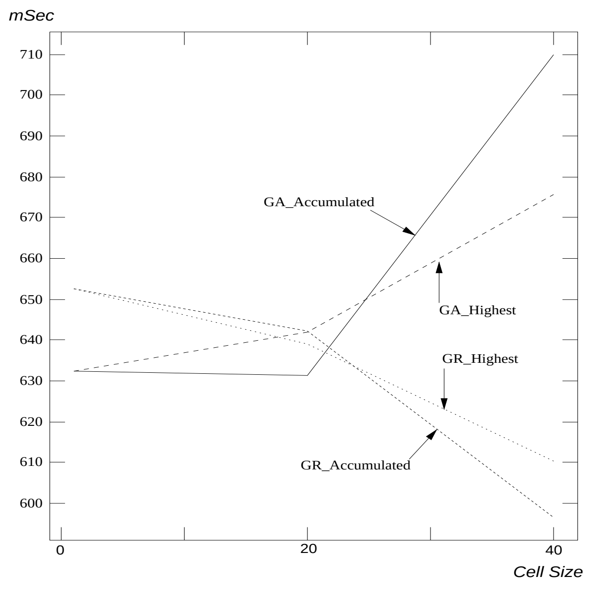


(b) Average Transaction Turnaround Times

Figure 19: Sensitivity Analysis: Experimental Set 1 – Workload $W3$



(a) Average Object Response Times

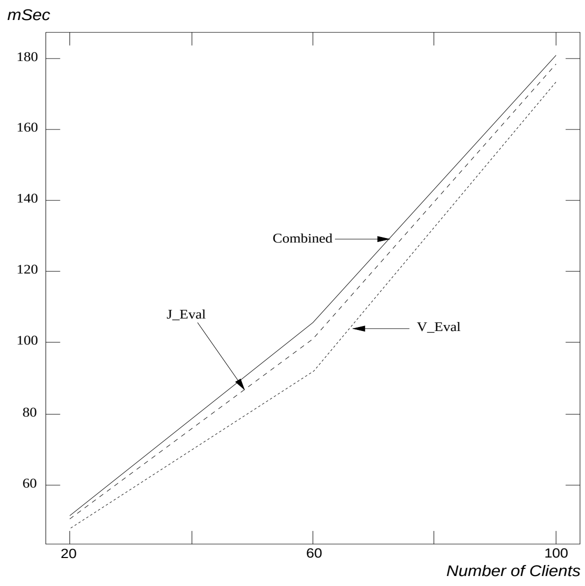


(b) Average Transaction Turnaround Times

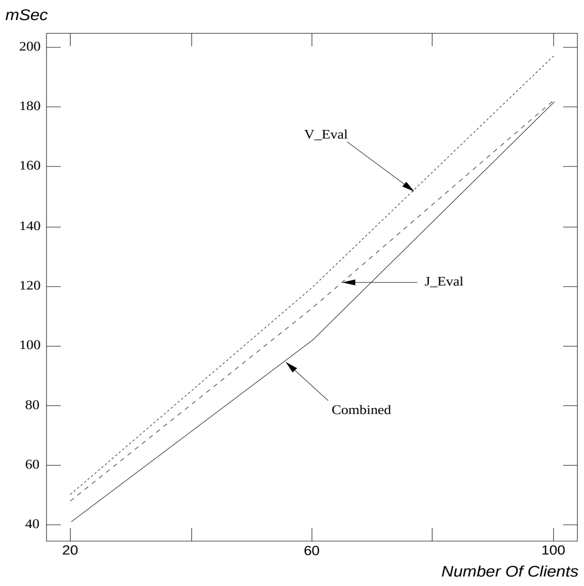
Figure 20: Sensitivity Analysis: Experimental Set 1 – Workload $W4$

only the latter for this set of experiments. First, we used only the J evaluation metric to guide the (Equation 1) genetic algorithm, then we used the V metric (Equation 2), and finally we used a combination of these two metrics (Equation 3). In computing the combined metric, the value used for α was 3 for 20 clients, and 35 for greater than 20 clients. The value of β was 1. These values were chosen so as to make the contributions of J and V be of the same order of magnitude. In the access data pre-processing phase, we used a 10:1 mapping, i.e., ten database objects were mapped to each cell. The cell-cumulative technique was used to represent each cell.

The object response times in the 3t-CSD system configurations created using the three different evaluation metrics are shown in Figure 21(a). The workload used here is $W3$. As the graph depicts, the average response times measured in the 3t-CSD for the three different evaluation metrics are very close. Using the V evaluation metrics alone in the genetic algorithm (to judge the quality of clustering solutions) results in a slightly better performance than the other two. This is because in a randomized hot spot scenario like $W3$, it is difficult to optimally identify data access overlapped clusters. In such situations, demarcating clusters such that inter-cluster data accesses are minimized is easier, and therefore tends to offer better clustering solutions and, therefore, shorter response times for object requests.



(a) Average Object Response Times for Workload $W3$



(b) Average Object Response Times for Workload $W4$

Figure 21: Sensitivity Analysis: Experimental Set 2

In contrast to the results obtained with the $W3$ workload, those for the $W4$ workload show that the average object response time for 3t-CSD is lowest for the combined measure (Figure 21(b)). The reason why the J metric and the combined evaluation function lead to better clustering solutions is that, in the $W4$ workload, hot spots for groups of clients are chosen from distinct ranges. Therefore, the clients already demonstrate a great deal of object access overlap, and using maximal access overlap as a a cost-function produces a better client clustering. Using just the V evaluation metrics results in the worst performance demonstrated by the 3t-CSD architecture.

8 Conclusions

In client-server databases (CSDs), where a number of machines (clients) share data hosted by a common "server", it has been observed that this server becomes a performance bottleneck when the number of clients becomes large (40 or more, in our experiments). In order to resolve this scalability problem, we propose that the clients in such systems be grouped into clusters based on the similarity in their data access patterns. The resulting configuration is a three-tier architecture (3t-CSD). Each such cluster is managed by an Intermediate Cluster Manager (ICM) that co-ordinates collaborative data sharing among clients in the cluster. When clients that access the same data are grouped together, the data sharing within the cluster can be optimized. The latter can reduce the number of required interactions with the database server.

In this paper, we prove that the optimal client clustering is an NP-complete problem. We then propose two techniques for logically clustering database clients depending on their data access behavior. In the first, we use a greedy strategy that assigns each client to the best candidate cluster at that point. The second proposed method is a genetic algorithm that simulates a "survival of the fittest" evolution in order to derive viable client-clusters. The initial population for the genetic algorithm can be generated in many different ways. Here, we use two techniques to achieve this, namely: (i) a purely randomized method that creates the initial clustering by simply assigning each client to an arbitrarily selected cluster, and (ii) a heuristic-based approach that assigns each client to the cluster that matches its own access pattern most closely. In developing our genetic algorithm, we have devised a new encoding scheme to represent feasible clustering solutions and a pre-processing technique that reduces the volume of the observed object access patterns.

We have developed two fully distributed prototypes of the two and three-tier architectures. The 3t-CSD is able to avoid the performance bottleneck at the server. This is done by off-loading the concurrency control and object request satisfaction for entire client clusters from the server to the respective ICMs.

Our main experimental conclusions are:

- For HOTCOLD type of database workloads, the proposed logically clustered 3t-CSD architecture yields significant performance improvements over its more conventional two-tier CSDs.
- The genetic algorithm composes well-formed clusters when the size of the mapping cells is small. As this size increases, the loss of detail in the clients' data access patterns causes the quality of the clustering generated by the GA to deteriorate.
- Since the greedy algorithm allocates client to cluster based solely on their data access overlap, it is less likely to get confused when the level of detail in the clients' accesses is reduced. Therefore, the greedy algorithm demonstrates an improvement in its performance when the mapping granularity is large.

- The evaluation functions used to guide the evolution process in the GA do affect the performance of the 3t-CSD architecture. When the hot spots accessed by clients are randomly distributed over the database, the evaluation function V, which is a measure of inter-cluster data accesses yields a better performance. On the other hand, when clients access hot spots from distinct regions of the database the combined evaluation function and J are able to provide a better clustering solution, and hence, lower average object response times.
- Our experiments show that with the proper abstraction (i.e., bitmaps) GA-based algorithms can be used to provide plausible solutions to a computationally intractable problem even in light of massive input sizes.

Acknowledgments: we would like to thank Maor Avidor and the reviewers for their thoughtful comments. A preliminary report on this work appeared in [22].

References

[1] S. Banerjee and P. Chrysanthis. Data Sharing and Recovery in Gigabit-Networked Databases. In *Proceedings of the Fourth International Conference on Computer Communications and Networks*, Las Vegas, NV, September 1995.

[2] M. Blaze and R. Alonso. Dynamic Hierarchical Caching in Large-Scale Distributed File Systems. In *Proc. 12th International Conference On Distributed Computing Systems*, Yokohama, Japan, June 1992.

[3] M. Carey, M. Franklin, M. Livny, and E. Shekita. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *ACM SIGMOD Conference*, May 1991.

[4] I.S. Chu and M.S. Winslett. Choices in Database Workstation-Server Architecture. In *Proceedings of the 17th Annual International Computer Software and Applications Conference*, Phoenix, AZ, November 1993.

[5] R. Cooley, B. Mobasher, and J. Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems*, 1(1), February 1999.

[6] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[7] M. Dahlin, C. Mather, R. Wang, T. Anderson, and D. Patterson. A Quantitative Analysis of Cache Policies for Scalable Network File Systems. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, May 1994.

[8] A. Delis and N. Roussopoulos. Performance Comparison of Three Modern DBMS Architectures. *IEEE Transactions on Software Engineering*, 19(2):120–138, February 1993.

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Massachusetts, USA, 1989.

[10] D. Goldberg. Genetic and Evolutionary Algorithms Come of Age. *Communications of the ACM*, 37(3), March 1994.

[11] J. Grefenstette. Optimisation of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, January/February 1986.

[12] G. Harhalakis, C.-P. Lin, L. Mark, and P. Muro-Medrano. Implementation of Rule-Based Information Systems for Integrated Manufacturing. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):892–908, December 1994.

[13] A. Hurson, S. Pakzad, and J. Cheng. Object-Oriented Database Management Systems: Evolution and Performance Issues. *IEEE Computer*, 26(2), February 1993.

[14] R. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[15] A. Leff, P.S. Yu, and J.L. Wolf. Policies for Efficient Memory Utilization in a Remote Caching Architecture. In *the First International Conference on Parallel and Distributed Information Systems*, Miami Beach, FL, December 1991.

[16] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[17] W.J. McIver and R. King. Self-adaptive, On-line Reclustering of Complex Object Data. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, MN, May 1994.

[18] M. Merz, F. Griffel, T. Tu, S. Muller-Wilken, H. Weinreich, M. Boger, and W. Lamersdorf. Supporting Electronic Commerce Transactions with Contracting Services. 7(4):249–274, December 1998.

[19] C. Mohan and I. Narang. ARIES CSA: a Method for Database Recovery in Client-Server Architectures. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):55–66, June 1994.

[20] E. Panagos, A. Biliris, H. Jagadish, and R. Rastogi. Client-Based Logging for High Performance Distributed Architectures. In *Proceedings of the Twelfth Internation Conference on Data Engineering*, pages 344–351, New Orleans, LA, USA, February 1996.

[21] J.H. Park and A. Delis. The Effect of Clustering in Client-Caching Architectures. In *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998.

[22] J.H. Park, V. Kanitkar, A. Delis, and R. Uma. On the Use of Genetic Algorithms in Database Client Clustering. In *Proceedings of the 1999 IEEE International Conference on Tools with Artificial Intelligence*, Chicago, IL, November 1999.

[23] R. Polamraju and W.D. Potter. Databases for Engineering Applications. In *IEEE SOUTHEASTCON*, volume 2, 1991.

[24] N. Roussopoulos, L. Mark, T. Sellis, and C. Faloutsos. An Architecture for High Performance Engineering Information Systems. *IEEE Transactions on Software Engineering*, 17(1), January 1991.

[25] K. Salem, D. Barbara, and R. Lipton. Probabilistic Diagnosis of Hot Spots. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 30–39, Tempe, AZ, 1992.

[26] A. Sinha. Client–Server Computing. *Communications of ACM*, 35(7), July 1992.

[27] A.S. Tanenbaum. *Distributed Operating Systems*. Prentice Hall – Computer Science, New York, NY, 1995.

[28] N. Yoshida and R. Araki. Efficient Implementation of Distributed Genetic Algorithms on Network of Workstations. In *Proceedimgs of the Second International Symposium on Soft-Computing*, pages 336–340, September 1997.