# Declustering Databases on Heterogeneous Disk Systems*

Ling Tony Chen
Lawrence Berkeley Lab
Berkeley, CA 94720

Doron Rotem
Lawrence Berkeley Lab
Berkeley, CA 94720
and
Department of MIS
San Jose State University
San Jose, California

Sridhar Seshadri
Stern School of Business
New York University

## Abstract

Declustering is a well known strategy to achieve maximum I/O parallelism in multi-disk systems. Many declustering methods have been proposed for symmetrical disk systems, i.e, multi-disk systems in which all disks have the same speed and capacity. This work deals with the problem of adapting such declustering methods to work in heterogeneous environments. In such environments there are many types of disks and servers with a large range of speeds and capacities. We deal first with the case of perfectly declustered queries, i.e., queries which retrieve a fixed proportion of the answer from each disk.

We propose an algorithm which determines the fraction of the dataset which must be loaded on each disk. The algorithm may be tailored to find disk loading for minimal response time for a given database size, or to compute a system profile showing the optimal loading of the disks for all possible ranges of database sizes.

The methods proposed here are general and can be used in conjunction with most known symmetric declustering methods.

*The support of the Defense Advanced Research Projects Agency, as well as the support of the Department of Energy under contract DE-AC03-76SF00098 is gratefully acknowledged.



**Proceedings of the 21st VLDB Conference
Zurich, Switzerland, 1995**

## 1 Introduction

Declustering methods have gained a lot of attention recently as a technique to enhance I/O parallelism [1, 2, 4, 5, 3, 6]. The idea is to distribute the data among $n$ parallel disks so that data which is likely to be requested together by a query is allocated to different disks. The speed-up is achieved due to the fact that each disk has to access only a fraction of the answer. Proposed declustering methods differ from each other in the way in which they decompose the data among the disks. Methods based on hashing techniques, error-correcting codes, hilbert curves and lattice-based decomposition have all appeared in recent research literature.

In symmetrical systems where the disks have similar transfer rates and capacities, the maximum I/O speed-up is achieved if the response to a typical query is balanced across the disks, i.e. each of the $n$ disks accesses approximately the same fraction, $1/n$, of the answer. To our knowledge, all of the research work on declustering focuses on symmetrical systems and therefore each of the decomposition algorithms mentioned above allocates the same volume of data to each disk.

However, many applications have to use existing platforms and hardware configurations and therefore need to decluster the data over a network of heterogeneous disks and servers. The various disks in the system may differ in their transfer rates, seek times and their capacities. Furthermore, the servers to which these disks are connected may have different speeds. As we will show later, server speeds may affect our solution as a slower server may limit the effective com-

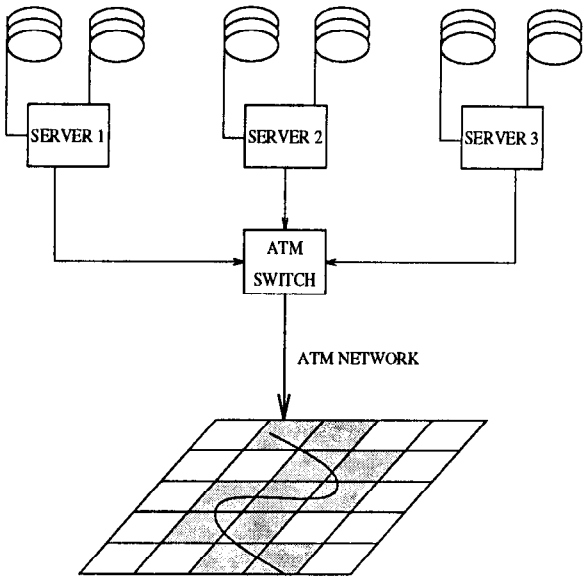


Figure 1: Declustering for terrain visualization application

bined transfer rate achievable by the disks connected to it.

In fact, this work was motivated by an application where we needed to design an Image Server in which we store a collection of aerial photographs of an area of interest for the purpose of terrain visualization. The photographs are partitioned into tiles of some standard size (typically 256 by 256 pixels) which are then declusterd on a system consisting of several servers each connected to a number of parallel disks. Users can trace a path of flight thru the terrain on the screen of a SGI workstation connected to the servers, all tiles intersecting this path must be fetched quickly from the disks to allow continuous visualization. Currently, the system runs on a network of three to six servers each connected to a number of disks with different capacities and speeds (See Figure 1). As an example of the kind of heterogeneity we are facing, the Image Server will be using a variety of disks such as the Seagate Barracuda and the Elite 2 whose speeds range between 1 to 3 MB/s, and servers ranging from a Sun Sparc 10 model 41 and 51 to Dec Alpha machines. The observed server speed ratio of the Sparc and the Alpha machines are approximately 1:2.

In [1] more details of the actual declusterization method are given. In developing declusterization strategies for the Image Server, we discovered that symmetric declusterization is not optimal due to the heterogeneity of our hardware devices and subsequently developed the general declustering algorithm described here.

In this paper we do not devise a new declusterization method but rather show how to adapt any declusterization method to work efficiently in heterogeneous environments. The method we propose here is especially suitable for declusterization of video and audio data as queries for such data types usually require large amounts of data that are accessed sequentially and can be easily declustered perfectly [7].

We present algorithms which determine the optimal ratio of data allocation to the various disks, taking into account all the above mentioned constraints. We then consider the realistic scenario in which the proportion of the data requested from each disk follows some multinomial distribution based on the fraction of the database stored on each disk.

The paper is organized as follows: In Section 2 we present some preliminary results on systems without any disk capacity constraints. In Section 3 we present our solution for maximizing system bandwidth subject to capacity constraints. In Section 4 we present our most general algorithm which shows how to implement all the previous results in an environment where disks are connected to different servers with their own bandwidth constraints. In Section 5 we analyze the case in which the request proportions follow some probabilistic distribution. Section 6 contains a discussion about the implementation of our algorithm and an example of the results produced by it on a typical system. Finally, Section 7 contains some conclusions and discussion of future work. Table 1 lists all the major notations used in this paper, most of which have not been introduced yet.

## 2 Declustering for maximal bandwidth without capacity constraints

In order to illustrate the concepts, let us consider a simple example in which we wish to store a dataset of size 1 GB on 2 disks with transfer rates of 3 and 2 MB/s respectively. Let us assume no capacity constraints are present (i.e. each disk has a capacity greater than 1 GB), and that future queries on this dataset request perfectly declustered chunks of data so that for every query the amount of data read off disk $i$ is proportional to the fraction of the dataset residing on disk $i$.

We note that the response time of a query is equal to the time it takes for the last byte to arrive, i.e., we need to wait for all disks to complete the transfer their portion of the answer. If we decluster this dataset symmetrically (i.e. 0.5 GB on each disk), a query requesting a chunk of 100 MB will retrieve 50MB from each disk. The response time for this query will be $\max(\frac{50}{3}, \frac{50}{2}) = 25$ seconds. On the other hand, if we allocate 0.6GB on the fast disk and 0.4GB on the slower one, the query will access (based on these pro-

| Notation | Meaning |
|---|---|
| $n$ | The number of disks in the System |
| $C_i$ | The usable capacity of disk $i$ |
| $a_i$ | The capacity actually stored in disk $i$ for this request |
| $B_i$ | The bandwidth (transfer rate) of disk $i$ |
| $C$ | The total amount of data requested to be stored |
| $B$ | The desired retrieval bandwidth |
| $T$ | The retrieval time constraint ($= C/B$) |
| $B_{opt}$ | The optimal (i.e. maximal) retrieval bandwidth |
| $T_{opt}$ | The optimal (i.e. minimal) retrieval time ($= C/B_{opt}$) |
| $m$ | The number of servers in the System |
| $S_j$ | The set of disks in server $j$ |
| $\mathcal{B}_j$ | The bandwidth of server $j$ |
| $\mathcal{C}_j$ | The current amount of data in server $j$ |

Table 1: List of all notations used in this paper

portions) on the average 60MB from the fast disk and 40MB from the slow disk resulting in an expected response time of $\max(\frac{60}{3}, \frac{40}{2}) = 20$ seconds.

Note that even in symmetric declustering schemes, it is quite possible that the actual observed response time for a query will be slower than the promised one due to the fact that the portion of the dataset requested by the query may not be perfectly declustered across the disks. In the non-symmetric case an additional complication may arise due to fluctuations in the ratios retrieved from each disk around their expected values. We will deal with these issues in Section 5.

In this section, we define a minimal response time declustering scheme to be a scheme which achieves minimal expected response time (i.e. maximal bandwidth) for **perfectly declustered queries**. Note that assuming all queries are perfectly declustered is especially true for video and audio data, where queries always access a continuous stream of data, and are almost always perfectly declustred. In the absence of capacity constraints, the above example illustrates the following principle summarized under proposition 1 (the proof is very simple and therefore omitted).

PROPOSITION 2.1 *Assuming that all disks are infinite in capacity, and that $C$ amount of data needs to be stored in a disk system containing $n$ disks with average transfer rates (i.e. bandwidths) of $B_1, B_2, \ldots, B_n$. The optimal declustering scheme that minimizes the response time for perfectly declustered query requests, should have the amount of data stored on each disk ($a_i$) be proportional to the bandwidth of each disk ($B_i$). In other words, the amount of data $a_i$ stored on disk $i$ should be:*

$$a_i = C \times \frac{B_i}{\sum_{k=1}^{n} B_k}$$

Note that this algorithm and the ones discussed in the following sections, will only determine a *declustering ratio* among the disks. In order to adapt a given symmetric declusterization strategy to follow the allocation ratios produced by our algorithm, we need to have each real disk be represented by multiple *virtual disks*. Considering the previous example where the determined declustering ratio is 3:2. We need to transform this initial problem into one with $3 + 2 = 5$ symmetric virtual disks and apply the symmetric declusterization method to a problem with 5 virtual disks. We then collect the data that has been declustered to 3 of these virtual disks, and load them into the first real disk, and collect the data from the other 2 virtual disks and load them into the second real disk. The question of how to choose the 3 virtual disks which should be loaded on the same real disk is application dependent and can usually be chosen to preserve maximum declusterization. Also note that in order to use the above method we need to have integer ratios, where as the declustering ratios determined by our algorithms are usually real numbers. This should not present a big problem though, since it is fairly easy to come up with good integer ratios based on real number ratios, with some small error introduced in the process.

# 3 Declustering for maximal bandwidth with capacity constraints

## 3.1 An example

Next we consider the more practical system in which each disk has some finite capacity of $C_i$, and discuss how to decluster with capacity constraints. As an example, consider the following problem: Assume we need to store a dataset of 2.5 GB in a system, with 3 disks, disk 1 can store 1 GigaByte and has a average transfer rate of 3 MB/s, disk 2 can store 2 GB and has a average transfer rate of 2 MB/s, and disk 3 can store 3 GB and has a average transfer rate of 1 MB/s. This system is illustrated in Figure 2. As we showed in the previous section, without considering disk capacity limitations, we should always decluster data on this disk system in a 3:2:1 ratio (proportional to the speed of the 3 disks). This would result in an average transfer rate of $3 + 2 + 1 = 6$ MB/s being observed by the whole system. The question is whether this transfer rate can be achieved with the capacity constraints we have introduced. As the following discussion shows

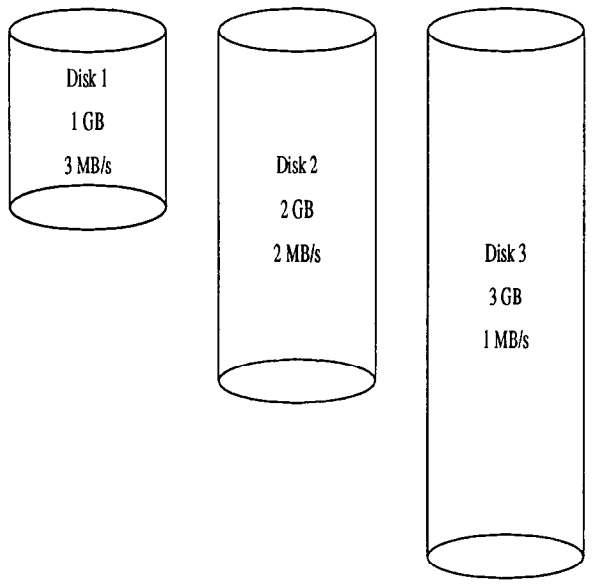


Figure 2: A disk system containing 3 heterogeneous disks

this rate is not achievable and finding the maximum achievable rate and optimal allocation is not trivial.

The first thing we need to realize is that in order for any subset to be retrieved at a transfer rate of 6 MB/s, we specifically need to be able to retrieve the entire dataset (2.5 GB) in 2500/6 = 416.66 seconds from all 3 disks, ever though most queries only ask for a portion of the dataset. By observing this constraint and the disk capacity constraints we note that we are allowed to place at most 1 GB on disk 1 (capacity constraint), 833.33 MB on disk 2 (transfer rate constraint) and 416.67 MB (transfer rate constraint) on disk 1. The total is less than 2.5 GB and thus 6 MB/s is not achievable, but this still leaves us clueless as to what kind of a distribution would result in a maximized retrieval bandwidth and what that bandwidth would be. For this particular problem, it turns out that the optimal solution is to place 1 GB on disk 1, 1 GB on disk 2, and 0.5 GB on disk 3, which would result in a maximized bandwidth of 5 MB/s. In the following subsection, we will show an algorithm that can help us find this optimal solution.

### 3.2 The algorithm

The problem in its more general form can be described as follows: We are required to load a dataset of $C$ MBytes on a system of $n$ disks with each disk $i$ having a capacity of $C_i$ MB and a transfer rate $B_i$ MB/s, such that the overall bandwidth $B$ of the system is maximized.

We note that the bandwidth of the system is a function of the individual transfer rates of the disks and the amount of data each of them needs to transfer. Clearly, under any feasible solution, the volume of data ,$a_i$, allocated to each disk should satisfy:

$$\forall i, 0 \leq a_i \leq C_i$$

$$C = \sum_{i=1}^{n} a_i$$

First let us consider how we can determine whether a given level of system bandwidth $B$ is achievable. In order for this bandwidth to be achieved, the entire dataset of size $C$ must be retrievable in $T = C/B$ time. This implies that disk $i$ should hold at most $T \times B_i$ MB of data. Of course, we also have the constraint that at most $C_i$ MB of data can be put on disk $i$. Thus, we define the function:

$$f(i, T) = \min(T \times B_i, C_i)$$

which indicates the maximum amount of data that can be stored on disk $i$, subject to the retrieval time constraint of $T$.

Define the total volume of data that can be stored on the system subject to the time constraint $T$ as

$$g(T) = \sum_{i=1}^{n} f(i, T)$$

PROPOSITION 3.1 *The maximal bandwidth, $B_{opt}$, achievable in the system satisfies:*

$$C = g(T_{opt})$$

*where*

$$T_{opt} = \frac{C}{B_{opt}}$$

PROOF: Let $B_{opt}$ be the maximal bandwidth we are looking for, and $T_{opt} = C/B_{opt}$. If $g(T_{opt})$ is larger than $C$, it would mean that an allocation could still be found with $B$ larger than $B_{opt}$ thus implying that $B_{opt}$ is not the maximal bandwidth. If $g(T_{opt})$ is smaller than $C$, it would mean it is infeasible. Thus it must be true that $g(T_{opt}) = C$ □

Due to the fact that $g$ contains a function $f$ which uses the min function, an inverse function for $g$ does not exist, and thus a closed form solution for $B_{opt}$ does not exist. The most efficient way to find $B_{opt}$, would be to perform a binary search on different bandwidths $B$. For each bandwidth $B$ that we try, if $g(C/B) > C$, then $B$ is too small and needs to be increased, if $g(C/B) < C$, then $B$ is too large and needs to be decreased. We keep on iterating this search until $g(C/B)$ is sufficiently close to $C$, at which point $B$ will be sufficiently close to $B_{opt}$. At this termination time, we

simply set the amount of data, $a_i$, we store in each disk to be $f(i, C/B)$.

The MaxBandwidth-Algorithm listed below contains a more formal description of the above algorithm.

**MaxBandwidth-Algorithm:**

1. Compute $B_{max}$, the highest possible bandwidth as $B_{max} = \sum_{i=1}^{n} B_i$
2. Using a binary search procedure between 0 and $B_{max}$ find a value $B$ that satisfies $0 \leq |g(T) - C| \leq \epsilon$ where $\epsilon$ is some predetermined acceptable small error, and $T = C/B$. At this point, we should also have $|B - B_{opt}| < \delta$ for some small $\delta$ that is a function of $\epsilon$.
3. Load the $i^{th}$ disk with $a_i$ equal to $f(i, C/B)$

As a final note about this algorithm we note that theoretically speaking, we can terminate the binary search, whenever the remaining range of possible $B$'s to search for, is such that, whether $f(i, T)$ is $T \times B_i$ or $C_i$ is completely determined for all $i$. At this point, a closed form solution can be obtained since an inverse function for $f$ exists for this range of $B$ on all $i$. From a practical point of view, however, the implementation is much simplified by letting the binary search continue to the point where the desired precision of $B$ is obtained. The complexity of the algorithm is $O(n \log_2 \frac{B_{max}}{\delta})$ which is really not that important, since the running time of this algorithm is negligible (less than a second) compared to the time it takes to load a dataset into a disk system.

### 3.3 Optimal loading for varying database sizes

In some cases the database size is not known apriori and the designer needs a "System Profile" which indicates how to optimally load the data on the disk system for each feasible database size. We can compute an optimal loading scheme for varying database sizes by using the following procedure whose correctness is based directly on Proposition 2.1. The procedure uses iterations where in each iteration more the amount of data loaded increases while the system transfer rate decreases. This continues until we fill all the capacities of our disks.

Given any portion of the database, $X$, which is still unloaded on the disks, and given a subset $U$ consisting of $k$ disks (of the $n$ initial disks) with remaining unused capacity, we will try to load as much as we can on $U$ observing the "proportionality" principle which says that the fraction of data loaded on each disk of $U$ must be proportional to the speed of that disk (within $U$). The system transfer rate achieved for the loaded portion is $\sum_{i \in U} B_i$. Observing the "proportionality" principle, we may not be able to load all of $X$ due to one or more disks reaching their capacity $C_l$, these are what we call the bottleneck disks. Each iteration is completed when one or more bottleneck disks are identified.

We note that by the above principle, if we load a database of size $X$ on the system, the amount loaded on the $i^{th}$ disk must satisfy $\frac{B_i}{B_{max}} \times X \leq C_i$ where $B_{max} = \sum_{i \in U} B_i$. From this it follows that $X \leq \frac{C_i}{B_i} \times B_{max}$ for all $i$. Therefore if the $l^th$ disk is a bottleneck it must satisfy $\frac{C_l}{B_l} = Min_{i \in U}\{C_i/B_i\}$.

At this point we iterate the procedure with $U$ consisting of the remaining non-bottleneck disks and $X$ consisting of the remaining unloaded portion of the database. The remaining capacity of each disk is of course adjusted with the amount loaded on it in the previous round.

More formally this is described in the following algorithm.

**MaxBandwidth-Algorithm with varying Database:**

Given a databse of size $X$, a set of disks $U$

Do While $X > 0$, and at least one $C_i > 0$ in $U$

1. Compute $B_{max}$ as $B_{max} = \sum_{i \in U} B_i$
2. Compute $\frac{C_i}{B_i}$ for each disk in $U$
3. Find $Min_i\{C_i/B_i\}$, call it $Min(U)$, and the disks which achieve it $BottleNeck(U)$
4. Load each disk in $U$ with the correct proportion of $Min(U) \times B_{max}$
5. Reduce each $C_i$ in $U$ to $C_i$ minus the allocated proportion for this disk
6. Set $U = U - Bottleneck(U)$ and $X = X - (|U| \times Min(U))$

EndDo

As an example assume, we have three disks with relative bandwidths 5,2,1 and relative capacities 2,4,3 (for simplicity we omit units as only the proportions are relevant). The maximal size database we can load is 9. The first disk becomes a bottleneck in the first iteration where we load 2, .8 and .4 on the disks respectively to maintain the proportions, the combined bandwidth is 8 and remaining capacities are 0,3.2 and 2.6. At this point 3.2 were loaded. In the next iteration, the two remaining disks must be loaded with proportions 2:1. We can show that 4.8 can now be loaded with one disk receiving 3.2 and the other 1.6. The combined bandwidth at this point drops to 3. We finally load the remaining 1 on the slowest disk at which point the system bandwidth drops to 1. The

bandwidth for any loaded database size between 0 to 9 can be computed using the above allocations.

We observe that all disks of the same type, i.e., same speed and capacity will become bottlenecks at the same iteration. Therefore, the algorithm will need in $O(n^2)$ time where $n$ is the number of different disk types in the system.

## 4 Server Bandwidth

In this section, we discuss the complicated issues that arise when the disks are distributed among multiple servers that have bandwidth restrictions themselves. The assumption is that each disk is located within some server, and each server (which can contain multiple disks) has a limitation on its retrieval bandwidth, possibly because of limited bandwidth on its bus, memory, or even CPU. One can view this problem as just adding another layer of bandwidth restrictions on groups of disks. The extra layer does not necessarily have to come from the existence of servers. It could also come from bandwidth limitations on the disk controller, system bus, or network hub. In fact, it is quite possible that within a disk system, there are several layers of bandwidth restrictions that are imposed on the disks in a hierarchical form.

In this paper we will only discuss how to handle one extra layer, and we will use the term *server* to represent the reason for the extra layer. The reader should keep in mind that this solution can be easily extended for multiple layers and that the extra layer does not have to be due to the existence of servers.

Let us assume that the $n$ disks are connected to $m$ servers, and that $S_j$ represents the set of disks connected to server $j$. Let $\mathcal{B}_j$ represent the bandwidth of server $j$, and define $\mathcal{C}_j$ to be the total capacity that we are attempting to place on all disks in $S_j$ (i.e. $\mathcal{C}_j = \sum_{i \in S_j} a_i$). If we now try to come up with an allocation that achieves a retrieval bandwidth of $B$ (and thus a retrieval time of $T = C/B$) by putting $a_i = f(i,T)$ amount of data on disk $i$, we could run into the problem of $\mathcal{C}_j$ being larger than $T \times \mathcal{B}_j$. This means that although the amount of data stored on each disk in server $j$ can be retrieved in time $T$, server $j$ is overflowed with data and cannot keep up with this data in order to deliver it all in time $T$.

The solution to this problem is to place a total of only $T \times \mathcal{B}_j$ data on all the disks in server $j$, and to do this for every server $j$ that overflows. With this solution in mind, we can now redefine the function $g(T)$ that describes the total amount of data that can be placed in the system, subject to the retrieval time constraint of $T$:

$$g(T) = \sum_{j=1}^{m} s(j,T)$$

where $s(j,T)$ is the amount of data that can be stored in server $j$, and is defined as:

$$s(j,T) = \min(T \times \mathcal{B}_j, \sum_{i \in S_j} f(i,T))$$

where $f(i,T)$ is the amount of data that can be stored in disk $i$ as defined in the previous section. With this new definition of $g$, we can now compute the amount of data that can be placed in a system, subject to the retrieval time constraint of $T$ for both disks and servers. Thus, the binary search algorithm described in the previous section, can still be applied to find the maximal bandwidth $B_{opt}$ in which $g(C/B_{opt}) = C$.

## 5 Probabilistic analysis of heterogeneous declustering

In this section we describe the declustering process where we remove the perfect declustering assumptions. Assuming that we load $N$ records on $n$ disks, where $a_i$ records are loaded on the $i^{th}$ disk, we note that, if $X_i$ is the (random) number of records requested from disk $i$ by a random transaction, then $X = (X_1, X_2, ..., X_n)$ has a multinomial distribution with parameters $(a_1/N, a_2/N, ..., a_n/N)$.

In order to compare different allocations to each other we will need some definitions from the theory of Majorization.

Definitions ([8])

Notation: Given a vector $a = (a_1, a_2, ..., a_n)$ , rearrange the components of this vector in decreasing order and denote the rearranged vector as $(a_{[1]}, a_{[2]}, ..., a_{[n]})$, with $a_{[1]} \geq a_{[2]} \geq ... \geq a_{[n]}$.

Majorization: Given two vectors $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_n)$, the vector $a$ is said to be majorized by the vector $b$, written as $a <_m b$ if:

$\sum_{i=1}^{k} a_{[i]} \leq \sum_{i=1}^{k} b_{[i]}$, $k = 1, 2, ..., n-1$

and $\sum_{i=1}^{n} a_{[i]} = \sum_{i=1}^{n} b_{[i]}$

Schur Concave Function: A real valued function $f$ defined on a set A $\subset R^n$ is Schur Concave if $a <_m b$ on A implies $f(a) \leq f(b)$.

Smaller in Usual Stochastic Order: A random variable $X$ is said to be smaller in the usual stochastic order than another random variable $Y$ if for all $t$, $P(X \leq t) \geq P(Y \leq t)$. This ordering will be written as $X \leq_{st} Y$. A consequence of this ordering is that, if $X \leq_{st} Y$ , then $E(f(X)) \leq E(f(Y))$ for all non-decreasing real valued functions $f$ and when the epectations exist.

PROPOSITION 5.1
*([8] ,p. 306) If $X = (X_1, X_2, ..., X_n)$ has a multinomial distribution with parameter $\theta = (\theta_1, \theta_2, ..., \theta_n)$, then $P_\theta\{s \leq X_i \leq t\}$ is a Schur-concave function of $\theta, -\infty \leq s \leq t \leq \infty$.*

We will first deal with disks with equal speeds and different capacities.

PROPOSITION 5.2 *If all disks have equal speeds and capacities $C_1 \geq C_2 ... \geq C_n$, and there are a total of $N$ records to be placed, with $N \leq \sum_{i=1}^{n} C_i$, then the retrieval time is stochastically minimized by placing $a_n = \min\{C_n, N/n\}$ in disk $n$, $a_{n-1} = \min\{C_{n-1}, (N - a_n)/(n-1)\}$ records in disk $(n-1)$, ..., $a_j = \min\{C_j, (N - \sum_{i=j+1}^{n} a_i)/(n-j+1)\}$ in disk $j$, ..., $a_1 = (N - \sum_{i=2}^{n} a_i)$ records on the first disk.*

PROOF: Given any allocation of records $b = (b_1, b_2, ..., b_n)$, rearrange the given vector $b$ in decreasing order and denote the rearranged vector as $(b_{[1]}, b_{[2]}, ..., b_{[n]})$. We claim that: $a_1 \leq b_{[1]}, a_1 + a_2 \leq b_{[1]} + b_{[2]}, ..., a_1 + a_2 + ... + a_i \leq b_{[1]} + b_{[2]} + ... + b_{[i]}$ and $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$. Because the same number of records are allocated, $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i = N$. If $C_n > N/n$, then $a_i = N/n, i = 1, 2, 3, ..$ and the claim follows immediately. Else, $a_n = C_n$ (by definition). Let $\rho = \sum_{i=1}^{n-1} a_i / \sum_{i=1}^{n} b_i \leq 1$. Let $b_{[i]} = \rho b_{[i]}, i = 1, 2, ..., n-1$ and apply the same reasoning to the vectors, $(a_1, a_2, ..a_{n-1})$ and $(b_{[1]}, b_{[2]}, ..b_{[n-1]})$. The claim follows by repeated appication of this inductive argument.

This shows that the vector $a <_m b$ (see definition). As mentioned above, if $X_i$ is the (random) number of records requested from disk $i$, then $X = (X_1, X_2, ..., X_n)$ has a multinomial distribution with parameters, $(a_1/N, a_2/N, ..., a_n/N)$ and $(b_1/N, b_2/N, ..., b_n/N)$ under the two arrangements. Therefore it follows from the above proposition, and the definition of a Schur-concave function that

$P_{(a_1/N, a_2/N, ..., a_n/N)}\{X_i \leq t\} \geq P_{(b_1/N, b_2/N, ..., b_n/N)}\{X_i \leq t\}, -\infty \leq t \leq \infty$

This satisfies the definition of stochastic minimization. □

It appears at first glance that the problem with unequal disk speeds can be solved by allocating records directly in proportion to the disk speeds. Unfortunately this need always be the case as shown in our analysis and simulation results given below. We can prove that allocating more records to a faster disk is stochastically optimal (the proof is very tedious and omitted here). Here we present two results, the first shows that the "intuitive" allocation of records in proportion to the disk speeds is asymptotically optimal for large request sizes. The second gives an approximate allocation rule.

PROPOSITION 5.3 *When there are $n$ disks with transfer rates $B_i, i = 1, 2, ..., n$, then the allocation of the fraction $B_i/(B_1 + B_2 + ... + B_n)$ of records to disk $i$ is asymptotically optimal as the size of the request $N$ increases.*

PROOF: The proof is given for n=2. Assume that the allocation of $p$ fraction of records has been made to the first disk. Given that the total number of records requested is $N$, the distribution of the number of records, $X$, requested from this disk has mean $= Np$ and standard deviation $= \sqrt{Np(1-p)}$. The distribution of the number of records requested from the second disk has mean $N(1-p)$ but the same standard deviation. Assume that $p/B_1 - (1-p)/B_2 = \epsilon > 0$. Then we note that:

$(Np/B_1 - N(1-p)/B_2)/\sqrt{N} = \epsilon\sqrt{N}$ and $p = \frac{B_1}{B_1+B_2} + \frac{B_1 B_2}{B_1+B_2}\epsilon$

From the first of these relations, the difference in means of the two random variables, $X$ and $(N - X)$ grows asymptotically with $N$. This implies that the maximum of the two random variables $X/(B_1\sqrt{N})$ and $(N-X)/(B_2\sqrt{N})$ asymptotically coincides with $X/(B_1\sqrt{N})$ (a rigorous proof of this fact is omitted, but the logic is that both these random variables have the same finite standard deviation which does not grow with $N$ whereas their means grow apart at the rate of $\sqrt{N}$). This in turn implies that:

$\frac{E(\max(X/B_1, (N-X)/B_2)}{\sqrt{N}} \rightarrow \frac{E(X/B_1)}{\sqrt{N}} = \frac{\sqrt{N}p}{B_1} = \frac{\sqrt{N}}{B_1+B_2} + \frac{B_2}{B_1+B_2}\epsilon$

Thus the difference in the allocation, $\epsilon$, must be made as small as possible. □

### 5.1 An approximate result

In this section we use the normal approximation to the binomial. We first derive an exact formula for optimally allocating data on two disks of different speeds. We then use a simpler formula to derive a general heuristic for allocating data on $n$ disks.

Consider a random variable, $X$, distributed Normally with mean $Np$ and standard deviation $\sqrt{Np(1-p)}$. Then,

$E(\max\{X/B_1, (N-X)/B_2\}) =$

$E(X/B_1 I\{X \geq \frac{NB_1}{(B_1+B_2)}\}) + E((N-X)/B_2 I\{X \leq \frac{NB_1}{(B_1+B_2)}\}) =$

$E(\left(X - \frac{NB_1}{(B_1+B_2)}\right) I\{X \geq \frac{NB_1}{(B_1+B_2)}\})/B_1 + \frac{N}{(B_1+B_2)} P(X \geq \frac{NB_1}{(B_1+B_2)})$

$+E(\left(\frac{NB_1}{(B_1+B_2)} - X\right) I\{X \leq \frac{NB_1}{(B_1+B_2)}\})/B_2 + (\frac{N}{B_2} - \frac{NB_1}{(B_1+B_2)B_2}) P(X \leq \frac{NB_1}{(B_1+B_2)}) = \frac{N}{(B_1+B_2)} + E(X - \frac{NB_1}{(B_1+B_2)}) I(X \geq \frac{NB_1}{(B_1+B_2)})/B_1$

$$+E(\left(\frac{NB_1}{(B_1+B_2)} - X\right) I\{X \leq \frac{NB_1}{(B_1+B_2)}\})/B_2 \quad (1)$$

where $I\{A\}$ is the indicator function of the set $A$. Let $\Phi, \Phi^c$ stand for the standard Normal distribution function and its complement. Let $\phi$ denote the standard normal density function. Let $\alpha = NB_1/(B_1+B_2)$, $\mu = Np$, and $\sigma = \sqrt{Np(1-p)}$. Then the right hand side of (9) can be simplified as:

$\frac{N}{(B_1+B_2)} + \frac{1}{B_1} \int_{\frac{NB_1}{(B_1+B_2)}}^{\infty} (X - \frac{NB_1}{(B_1+B_2)}) \phi(\frac{(X-\mu)}{\sigma}) dX +$

$\frac{1}{B_2} \int_{-\infty}^{\frac{NB_1}{(B_1+B_2)}} (\frac{NB_1}{(B_1+B_2)} - X) \phi(\frac{(X-\mu)}{\sigma}) dX =$

$\frac{N}{(B_1+B_2)} + \frac{1}{B_1} \int_{\frac{NB_1}{(B_1+B_2)}}^{\infty} (X - \mu) \phi(\frac{(X-\mu)}{\sigma}) dX -$

$\frac{1}{B_2} \int_{-\infty}^{\frac{NB_1}{(B_1+B_2)}} (X - \mu) \phi(\frac{(X-\mu)}{\sigma}) dX$

$+(\mu - \frac{NB_1}{(B_1+B_2)}) \Phi^C(\frac{\alpha-\mu}{\sigma})/B_1 - (\mu - \frac{NB_1}{(B_1+B_2)}) \Phi(\frac{\alpha-\mu}{\sigma})/B_2 =$

$\frac{N}{(B_1+B_2)} + (\mu - \frac{NB_1}{(B_1+B_2)}) \Phi^C(\frac{\alpha-\mu}{\sigma})/B_1 - (\mu - \frac{NB_1}{(B_1+B_2)}) \Phi(\frac{\alpha-\mu}{\sigma})/B_2$

$-\frac{\sigma}{B_1} \int_{\frac{NB_1}{(B_1+B_2)}}^{\infty} d\phi(\frac{(X-\mu)}{\sigma}) + \frac{\sigma}{B_2} \int_{-\infty}^{\frac{NB_1}{(B_1+B_2)}} d\phi(\frac{(X-\mu)}{\sigma}) =$

$$N/(B_1+B_2) + (\mu - \alpha)(-\Phi(\frac{\alpha-\mu}{\sigma})/B_2$$

$$+\Phi^c(\frac{\alpha-\mu}{\sigma})/B_1) + \sigma\phi(\frac{\alpha-\mu}{\sigma})(\frac{1}{B_2} + \frac{1}{B_1}) \quad (2)$$

This expression can be minimized by using a search procedure. A approximate but quicker result can be obtained by noting that $\sigma$ is almost invariant for small changes in $p$. Then the expression to be minimized can be written as:

$-\frac{\alpha-\mu}{\sigma}(-\Phi(\frac{\alpha-\mu}{\sigma})/B_2 + \Phi^c(\frac{\alpha-\mu}{\sigma})/B_1) + \phi(\frac{\alpha-\mu}{\sigma})(\frac{1}{B_2} + \frac{1}{B_1})$

$$= -z(-\Phi(z)/B_2 + \Phi^c(z)/B_1) + \phi(z)(\frac{1}{B_2} + \frac{1}{B_1}) = H(z) \quad (3)$$

where $z$ is the standard Normal deviate.

$$\frac{dH(z)}{dz} = (\Phi(z)/B_2 - \Phi^c(z)/B_1) \quad (4)$$

and

$$\frac{d^2H(z)}{dz^2} = (\phi(z)/B_2 + \phi(z)/B_1) > 0 \quad (5)$$

Therefore $H(z)$ is minimized by setting the first derivative equal to zero, giving :

$$1/B_2 = \Phi^c(z)(1/B_1 + 1/B_2) \Leftrightarrow$$

$$\Phi^c\left(\frac{\alpha - Np}{\sqrt{Np(1-p)}}\right) = \left(\frac{B_1}{B_1+B_2}\right) \quad (6)$$

This is a "newsboy" type of solution. Equation 6 shows that when the disk speeds are equal we must have $\alpha = Np$, otherwise we must always favor the faster disk in allocating records, i.e., place more than the number proportional to its speed. We summarize these results:

PROPOSITION 5.4 *Given two disks with speeds $B_1$ and $B_2$ and a request of size $N$, then the optimal allocation under the Normal approximation to the binomial distribution is given by minimizing Equation (2). An approximate solution to this minimization problem can be obtained by solving for p in Equation 6 and placing p fraction of records on disk 1.*

Remarks: To apply this Proposition, first we need a value for $N$. We can use the average size of the request as a proxy for $N$. Second, when there are many disks, how can the allocation be made? We suggest that records are at first allocated in proportion to the disk speeds. Then iteratively reallocate records using (6) to the two fastest disks, the next two fastest and so on. Repeat this reallocation procedure until the changes in allocation are small.

The psuedo code for the allocation procedure is given below.

**Procedure HEURISTIC:**

Step 0:

Read number of disks (ndisk), disk speeds ($B_i$) and average number records retrieved (nrec).
Read tolerance (tol). We assume that disk speeds are ordered as: $B_1 > B_2 > B_3 .. > B_{ndisk}$
Set proportions to be allocated on $disk_i$ as $p_i = \frac{B_i}{(B_1+B_2+..+B_{ndisk})}$
Set error = 1.0e32

Step 1:
Do while $(error > tol)$
Step 1.1:
error = 0.0
Do i = 1, ndisk-1
$N = (p_i + p_{i+1}) * nrec$
$b_1 = B_i$
$b_2 = B_{i+1}$
Solve for p in (6) using a search procedure
$temp = p_i$
$p_i = p * N/nrec$
$p_{i+1} = (1-p) * N/nrec$
$error = error + |temp - p_i|$|
enddo
enddo

Usually the procedure converges in 10-20 iterations of Step 1.1. It can be proved that this procedure will eventually stop, (the reason is that the largest allocation, $p_1$, is monotone increasing). Our experiments show that the allocation is rather robust with respect to the average size of request N.

The allocation given by the procedure was tested against the proportional allocation using simulation. In the simulations, the average size of requests, N, was varied between 5 and 100 in steps of 5. For each average size N, 10,000 trials were conducted. In each trial, a random number $M$ was generated between 0.5*N and 1.5*N, where M is the actual size of the request. The results of these heuristics are shown in the next section.

The HEURISTIC procedure is easily adapted to the case when the disk capacities are finite. The HEURISTIC procedure is called and the allocations are tested for feasibility, i.e., whether the allocations can be fitted into the disks. If the allocation is feasible, we stop, else we look at the fastest disk whose allocation violates the capacity constraint. We load this disk to its capacity, eliminate it from further consideration and resolve the problem. The algorithm for handling this case is given below:

**Algorithm FINITE**

Step 0:
Read number of disks (ndisk), disk speeds $B_i$ and average number records retrieved (nrec).
Read tolerance $(tol)$. We assume that disk speeds are ordered as:$B_1 > B_2 > B_3.. > B_{ndisk}$
Read the capacity $c_i$, of each disk and the total number of records to be allocated, NTOT.
$alloc_i$ is the final allocation, $index_i$ is a temporary array, ntotold = NTOT
Set $index_i = i, i = 1, ndisk$
Step 1:
call HEURISTIC and get $p_i$
Step 2:

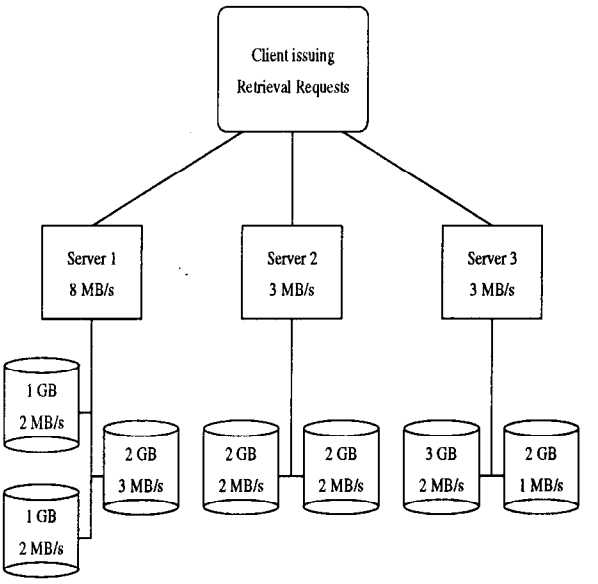


Figure 3: An example of a 3 server, 7 disk, heterogeneous system

Do for i = 1, ndisk
$alloc_{index_i} = p_i * NTOT/ntotold$
if$(p_i * NTOT > c_{index_i})$ then
$alloc_{index_i} = c_{index_i}/ntotold$
$NTOT = NTOT - c_{index_i}$
do for $j = i+1, ndisk$
$index_{j-1} = index_j$
enddo
$ndisk = ndisk - 1$
go to Step 1
endif
enddo
Stop

This iterative method can be used with any allocation method (i.e., not just HEURISTIC), by replacing the call in Step 1 of the algorithm to a call to the appropriate procedure. Using this logic, we compared the allocation under HEURISTIC versus allocation under the proportional scheme. Some graphs of the results are shown in the next section.

## 6 Implementation Results

The above described algorithms have all been implemented and tested to verify their correctness. As an example, Figure 4 shows what the maximal achievable bandwidth would be as a function of the load request size for the 3 server disk system shown in Figure 3. Server 1 of the system has a bandwidth of 8 MB/s and contains 3 disks, two of which are 1 GB disks with a bandwidth of 2 MB/s and the remaining one being a 2 GB disk with a bandwidth of 3 MB/s. Server 2 has

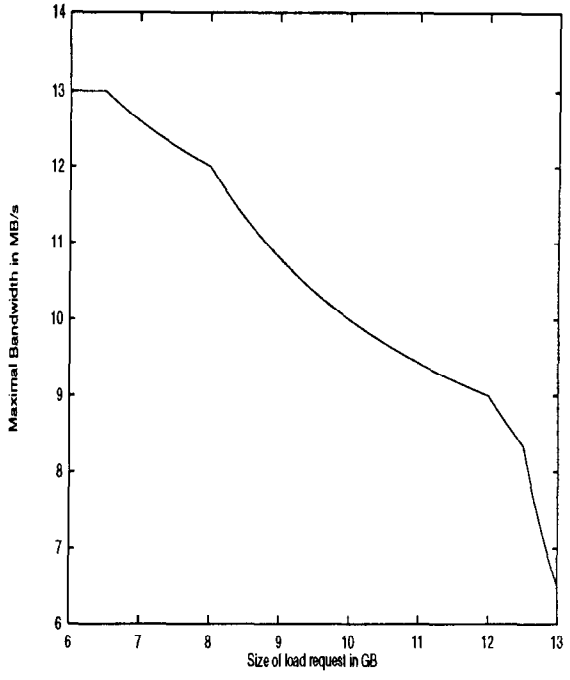


Figure 4: Achievable bandwidth relative to load request size

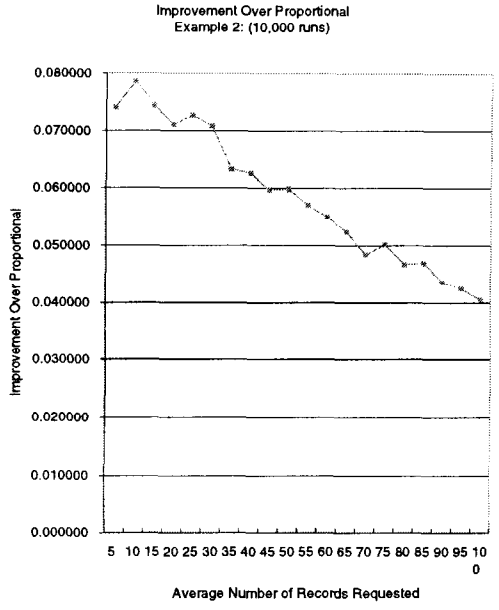


Figure 5: Disk speeds; 20,10,5,5

a bandwidth of 3 MB/s and contains 2 similar disks, each capable of holding 2 GB of data with a bandwidth of 2 MB/s. Server 3 also has a bandwidth of 3 MB/s but contains 2 different disks, one of which is a 3 GB disk with a bandwidth of 2 MB/s, and the other being a 2 GB disk with a bandwidth of 1 MB/s. In this disk system, only server 2 has a smaller bandwidth than the combined bandwidth of its disks. Because the two disks are of the same size and speed, the effective bandwidth of each disk turns out to be scaled down evenly to the point where they add up to the bandwidth of the server (i.e. 1.5 MB/s).

The maximal achievable bandwidth on this disk system (for small request sizes) is 13 MB/s (7 MB/s from server 1, and 3 MB/s each from server 2 and 3). When the size of the load request grows to 6.5 GB, the two 1 GB disks in server 1 get saturated and can no longer hold any more data. This is the reason for the sudden change in the slope of the curve at 6.5 GB. From 6.5 GB up to 13 GB (the full storage capacity of the system), the curve turns out to be piecewise hyperbola, with the intersection points between hyperbolas representing the saturation of other disks. In particular, at the storage capacity of 8 GB, the 2 GB disk in server 1 gets saturated, at 12 GB the two 2 GB disks in server 2 are saturated, and finally at 12.5 GB, the 3 GB disk of server 3 gets saturated. Of course, when we finally reach 13 GB, the only remaining slowest disk (the 2 GB disk of server 3) also gets saturated, and thus no more data can be placed in the system.

For the probabilistic method, we show graphs of the improvement in allocation resulting from our procedure as compared with simply using the deterministic methods. For the test cases shown the improvement ranges from 8 to 23 percent. In the graphs of Example 2 and Example 4 we show the comparison for disks with sufficient capacities. Example 2 uses 4 disks with relative speeds 20,10,5,5 whereas in Example 4 the speeds are 10,5,1,1. The graphs of Example 3 and Example 5 have finite capacities. The relative disk speeds in Example 3 are 20,15,10,7 with capacities 400,100,100,100 with a total database size of 600 ,in Example 5 disk speeds are 10,5,1,1 with capacities 750,200,100,100 with total database size of 1000.

## 7 Conclusion

In this paper we have described algorithms for adapting declusterization methods to work in heterogeneous distributed environments. The results reported here are particularly suitable for systems which store video and audio data on parallel disk systems but can be used for any environment in which declusterization is desirable such as image data or multidimensional data retrieved by range queries. The algorithms reported here were all implemented and are actually used in an

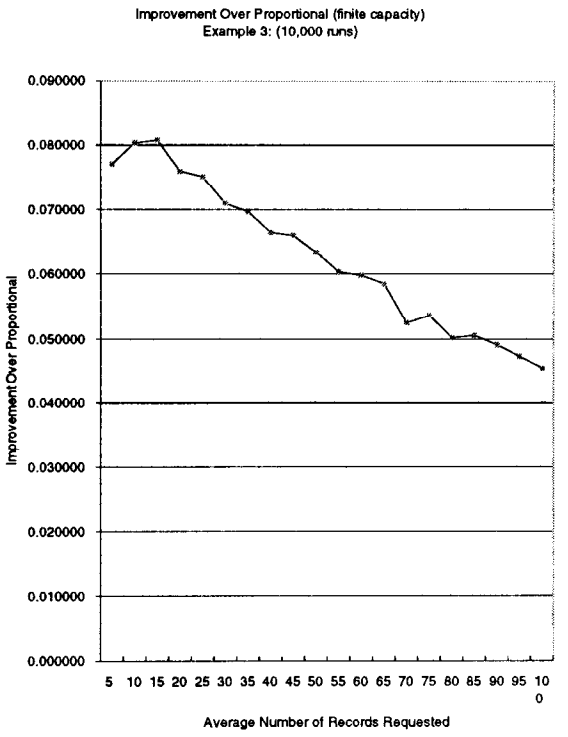


Figure 6: Disk speeds:20,15,10,7 with Capacities 400,100,100,100

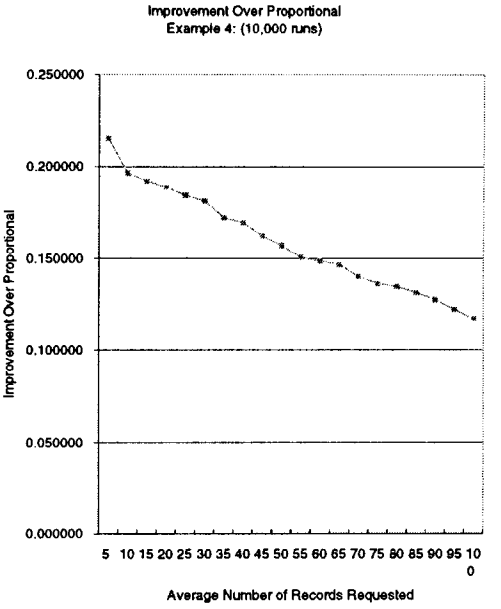


Figure 7: Disk speeds: 10,5,1,1

existing application which declusters image data for the purpose of terrain visualization.

The work reported here can also be used for system design purposes. As hardware configurations tend to be dynamic, designers may wish to replace disks and servers by newer available models. For example, a designer may wish to find which disks or servers in the current system should be upgraded in the most cost-effective way to improve system response time. Our algorithms provide as a by-product, information about the bottleneck disks and servers so that the designer can identify these components which are critical.

To summarize we have shown the following results:

- For perfectly declustered queries we have an exact algorithm to produce optimal allocations.
- We have provided a stochastically optimal allocation scheme for for finite capacity and same speed disks.
- We provide heuristic algorithms for disks with unequal speeds. The theoretical results show that we must allocate more than proportionally to faster disks. These ideas have been combined into a fast procedure for doing the allocation.
- Simulation results show that if disk speeds are more unequal, then greater benefit is derived from using the allocations from our procedure FINITE as opposed to the deterministic algorithm. Results also indicate that the benefit is greater when the size of the actual requests is small.

We plan to investigate the possibility of building design tools which will be able to select the most cost-effective products from a given list of alternative disks and server configurations. Such tools will employ these algorithms to determine the bandwidth of a selected system.

Another direction of future research involves some interesting probabilistic issues raised by this work, such as finding exact bounds on the probability that the response time will not deviate from the predicted one, based on the assumption that retrieval requests are completely random and cannot be perfectly declustered.

In the current work we deal with loading one dataset at a time on the system. Another issue we are currently exploring is that of finding optimal methods of loading multiple datasets on the disks where each dataset has its own desired bandwidth and capacity requirements as well as an associated *weight* that informs us how important it is to achieve the desired bandwidth (or come close to it).

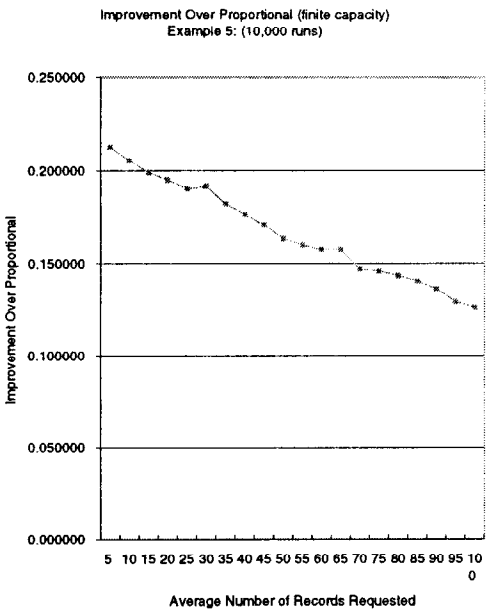


Figure 8: Disk Speeds: 10,5,1,1 with capacities 750,200,100,100

## References


[1] Ling Tony Chen and D. Rotem. "declustering objects for visualization". In *Proceedings of the 19th VLDB Conference*, pages 85–96, 1993.

[2] David J. DeWitt and S. Ghandeharizadeh. "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machine". In *Proc. 16th international Conference on VLDB*, pages 481–492, August 1990.

[3] H. C. Du. "Disk Allocation Methods for Binary Cartesian Product Files". *BIT*, 26:138–147, 1986.

[4] C. Faloutsos and P. Bhagwat. "Declustering Using Fractals". In *Second International Conference on Parallel and Distributed Computing (PDIS)*, pages 18–25, January 1992.

[5] F. Faloutsos and D. Metaxas. "Disk Allocation Methods Using Error Correcting Codes". *IEEE trans. on Computers*, 40(8):907–914, August 1991.

[6] S. Ghandeharizadeh, I. Ramos, Z. Asad, and W. Qureshi. "Object Placement in Parallel HyperMedia Systems". In *Proc. 17th international Conference on VLDB*, pages 243–254, September 1991.

[7] S. Ghandeharizadeh and L. Ramos. "Continuous retrieval of multimedia data using parallelism". *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–669, August, 1993.

[8] Marshall. A. W. and I. Olkin. *"Theory of Majorization and its Applications"*. Academic Press, Edinburgh, London, 1966.